bdiGDB

JTAG debug interface for GNU Debugger

ARMv8

User Manual

Manual Version 1.01 for BDI2000

abatron

©1997-2015 by Abatron AG

1 Introduction 4
1.1 BDI2000 4
1.2 BDI Configuration 5

2 Installation 6
2.1 Connecting the BDI2000 to Target 6
2.1.1 Serial Wire Debug 9
2.2 Connecting the BDI2000 to Power Supply 10
2.3 Status LED «MODE» 11
2.4 Connecting the BDI2000 to Host 12
2.4.1 Serial line communication 12
2.4.2 Ethernet communication 13
2.5 Installation of the Configuration Software 14
2.5.1 Configuration with a Linux / Unix host 15
2.5.2 Configuration with a Windows host 17
2.5.3 Recover procedure 18
2.6 Testing the BDI2000 to host connection 19
2.7 TFTP server for Windows 19

3 Using bdiGDB 20
3.1 Principle of operation 20
3.2 Configuration File 21
3.2.1 Part [INIT] 22
3.2.2 Part [TARGET] 24
3.2.3 Part [HOST] 30
3.2.4 Part [FLASH] 32
3.2.5 Part [REGS] 36
3.3 Debugging with GDB 38
3.3.1 Target setup 38
3.3.2 Connecting to the target 38
3.3.3 Breakpoint Handling 39
3.3.4 GDB monitor command 39
3.3.5 Target serial I/O via BDI 40
3.3.6 Target DCC I/O via BDI 41
3.4 Telnet Interface 42
3.4.1 Command list 43
3.5 Multi-Core Support 46

4 Specifications 49

5 Environmental notice 50

6 Declaration of Conformity (CE) 50

7 Abatron Warranty and Support Terms 51
7.1 Hardware 51
7.2 Software 51
7.3 Warranty and Disclaimer 51
7.4 Limitation of Liability 51

© Copyright 1997-2015 by ABATRON AG Switzerland

7.4 Appendices

A Troubleshooting 52

B Maintenance 53

C Trademarks 55

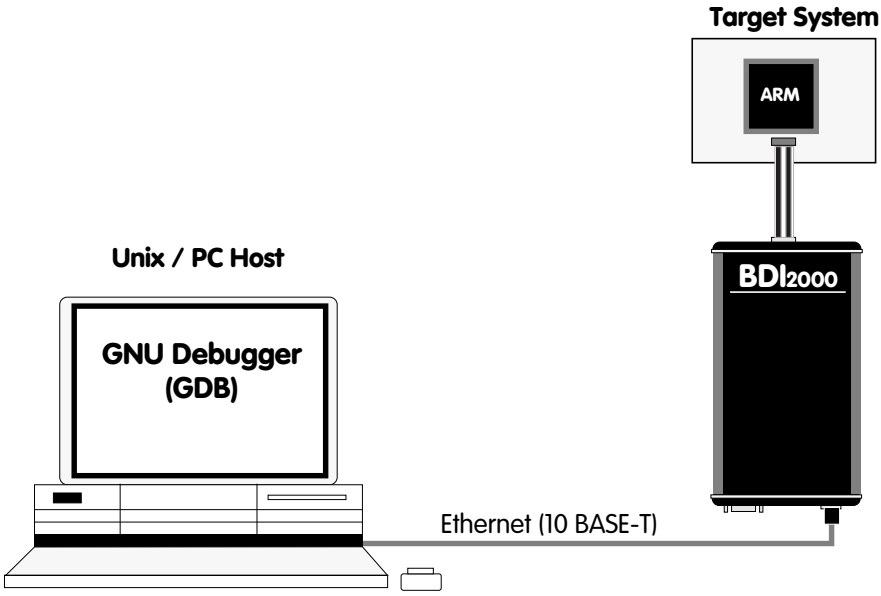
© Copyright 1997-2015 by ABATRON AG Switzerland

1 Introduction

bdiGDB enhances the GNU debugger (GDB), with JTAG debugging for ARMv8 based targets. With the built-in Ethernet interface you get a very fast download speed. No target communication channel (e.g. serial line) is wasted for debugging purposes. Even better, you can use fast Ethernet debugging with target systems without network capability. The host to BDI communication uses the standard GDB remote protocol.

An additional Telnet interface is available for special debug tasks (e.g. force a hardware reset, program flash memory).

The following figure shows how the BDI2000 interface is connected between the host and the target:



1.1 BDI2000

The BDI2000 is the main part of the bdiGDB system. This small box implements the interface between the JTAG pins of the target CPU and a 10Base-T ethernet connector. The firmware and the programable logic of the BDI2000 can be updated by the user with a simple Windows / Linux configuration program. The BDI2000 supports 1.8 – 5.0 Volts target systems (3.0 – 5.0 Volts target systems with Rev. A/B).

.

© Copyright 1997-2015 by ABATRON AG Switzerland

1.2 BDI Configuration

As an initial setup, the IP address of the BDI2000, the IP address of the host with the configuration file and the name of the configuration file is stored within the flash of the BDI2000.
Every time the BDI2000 is powered on, it reads the configuration file via TFTP.
Following an example of a typical configuration file:

```
; bdiGDB configuration for ARMv8 based X-Gene processor
; ------------------------------------------------------
;
[INIT]
; empty init list
;
[TARGET]
POWERUP         3000                    ;start delay after power-up detected in ms
CLOCK           8000000                 ;JTAG clock 8 MHz
TRST            OPENDRAIN               ;TRST driver type (OPENDRAIN | PUSHPULL)
;
; CoreID#0 parameters (active core after reset)
#0 CPUTYPE      X-GENE 0xfc010000       ;X-Gene CPU 0
#0 STARTUP      HALT                    ;halt as soon as possible
#0 ENDIAN       LITTLE                  ;memory model (LITTLE | BIG)
#0 VECTOR       CATCH  RST OSU TDA      ;Reset and OS unlock catch, Trap SW access
#0 BREAKMODE    SOFT                    ;SOFT or HARD
#0 MEMACCESS    CORE   10               ;memory access via Core  (80 TCK's access delay)
;
; CoreID#1 parameters:
#1 CPUTYPE      X-GENE 0xfc110000       ;X-Gene CPU 1
#1 STARTUP      RUN
#1 ENDIAN       LITTLE
#1 BREAKMODE    HARD
#1 MEMACCESS    CORE   10
;
; CoreID#2 parameters:
#2 CPUTYPE      X-GENE 0xfc210000       ;X-Gene CPU 2
#2 STARTUP      RUN
#2 ENDIAN       LITTLE
#2 BREAKMODE    HARD
#2 MEMACCESS    CORE   10
;
; CoreID#3 parameters:
#3 CPUTYPE      X-GENE 0xfc310000       ;X-Gene CPU 3
#3 STARTUP      RUN
#3 ENDIAN       LITTLE
#3 BREAKMODE    HARD
#3 MEMACCESS    CORE   10
;
;
[HOST]
#0 PROMPT       XGENE#0>
#1 PROMPT       XGENE#1>
#2 PROMPT       XGENE#2>
#3 PROMPT       XGENE#3>


[REGS]
FILE            $regARMV8-EL2.def
```

Based on the information in the configuration file, the target is automatically initialized after every reset.

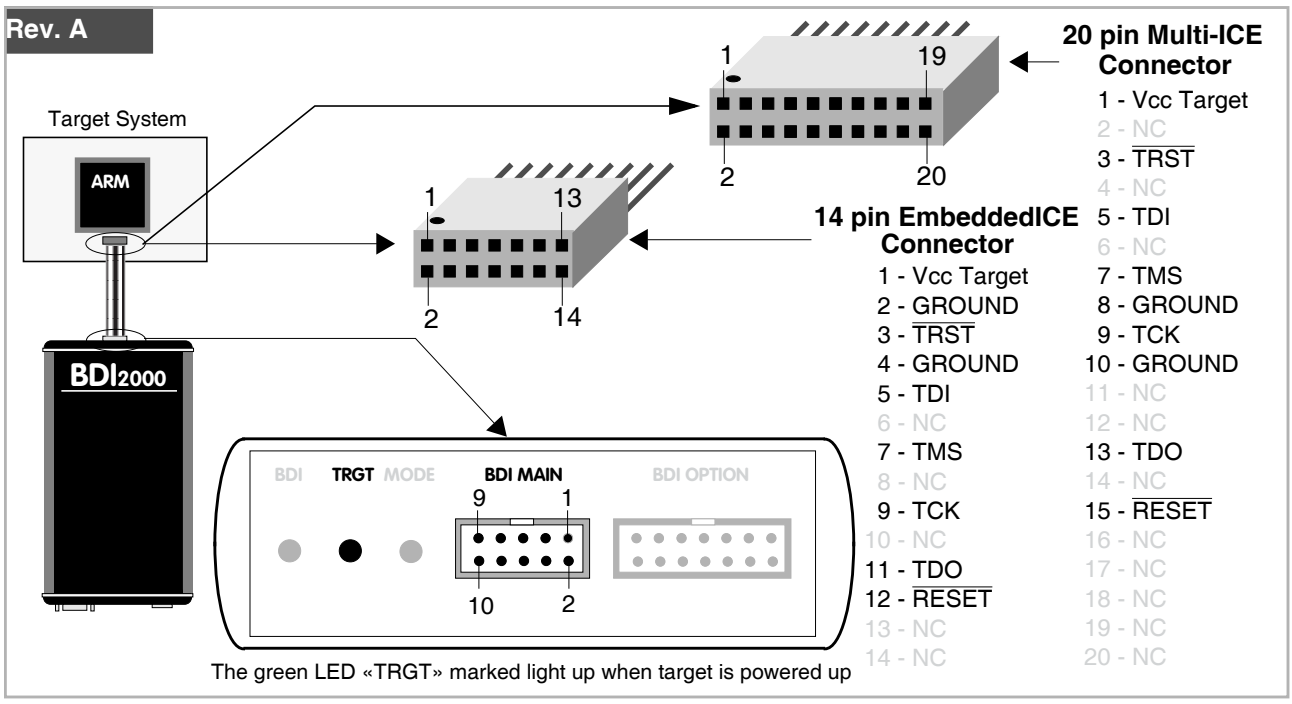
© Copyright 1997-2015 by ABATRON AG Switzerland

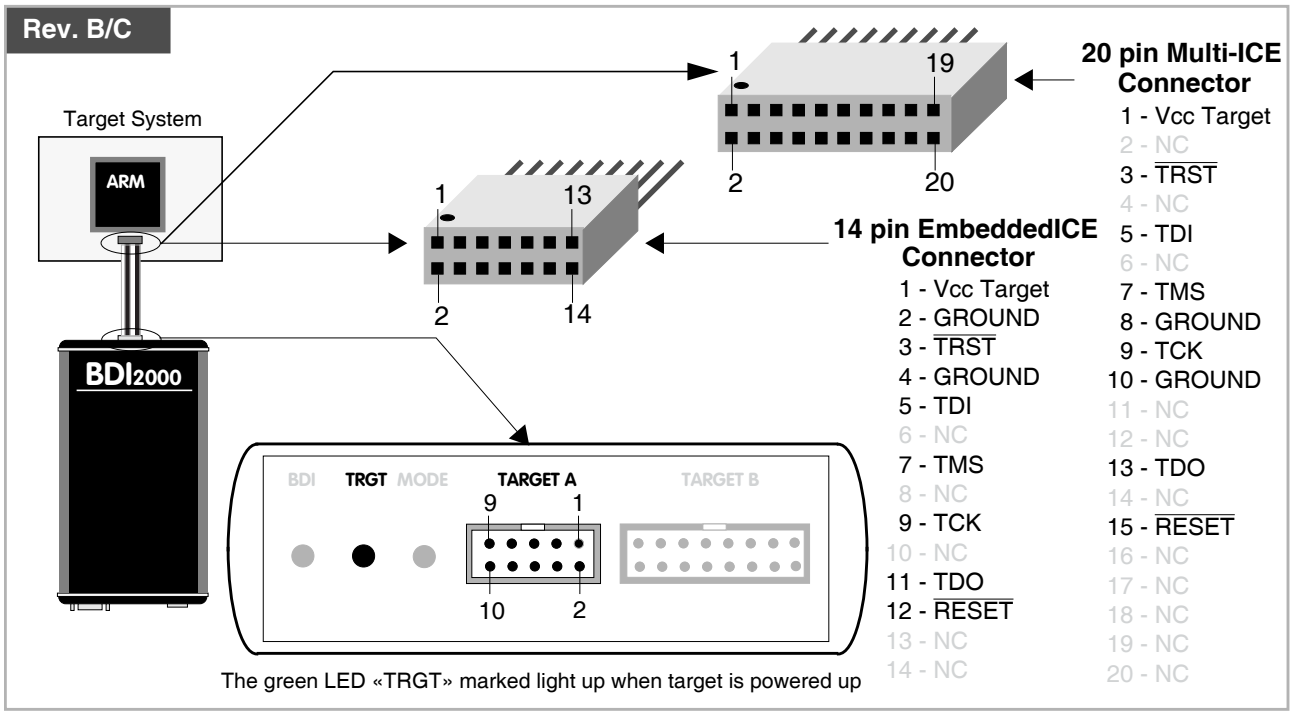
2 Installation

2.1 Connecting the BDI2000 to Target

The enclosed cables to the target system are designed for the ARM Development Boards. In case where the target system has the same connector layout, the cable can be directly connected (14-pin EmbeddedICE or 20-pin Multi-ICE).

In order to ensure reliable operation of the BDI (EMC, runtimes, etc.) the target cable length must not exceed 20 cm (8").





© Copyright 1997-2015 by ABATRON AG Switzerland

BDI MAIN / TARGET A Connector Signals

Pin	Name	Describtion
1	reserved	This pin is currently not used.
2	$\overline{\text{TRST}}$	**JTAG Test Reset** This open-drain / push-pull output of the BDI2000 resets the JTAG TAP controller on the target. Default driver type is open-drain.
3+5	GND	**System Ground**
4	TCK	**JTAG Test Clock** This output of the BDI2000 connects to the target TCK line.
6	TMS	**JTAG Test Mode Select** This output of the BDI2000 connects to the target TMS line.
7	$\overline{\text{RESET}}$	This open collector output of the BDI2000 is used to reset the target system.
8	TDI	**JTAG Test Data In** This output of the BDI2000 connects to the target TDI line.
9	Vcc Target	**1.8 – 5.0V:** This is the target reference voltage. It indicates that the target has power and it is also used to create the logic-level reference for the input comparators. It also controls the output logic levels to the target. It is normally fed from Vdd I/O on the target board. **3.0 – 5.0V with Rev. A/B :** This input to the BDI2000 is used to detect if the target is powered up. If there is a current limiting resistor between this pin and the target Vdd, it should be 100 Ohm or less.
10	TDO	**JTAG Test Data Out** This input to the BDI2000 connects to the target TDO line.

For TARGET B connector signals see table on next page.

The BDI2000 works also with targets which have no dedicated $\overline{\text{TRST}}$ pin. For this kind of targets, the BDI cannot force the target to debug mode immediately after reset. The target always begins execution of application code until the BDI has finished programming the Debug Control Register.

Note:
For targets with a **10-pin or 20-pin Cortex Debug Connector** (Samtec 0.05" micro header) a special adapter is available. This Cortex Adapter can be ordered separately from Abatron (p/n 90085).

Warning:
Before you can use the BDI3000 with an other target processor type (e.g. PPC <--> ARM), a new setup has to be done (see chapter 2.5). During this process the target cable must be disconnected from the target system.

To avoid data line conflicts, the BDI3000 must be disconnected from the target system while programming a new firmware for an other target CPU.

© Copyright 1997-2015 by ABATRON AG Switzerland

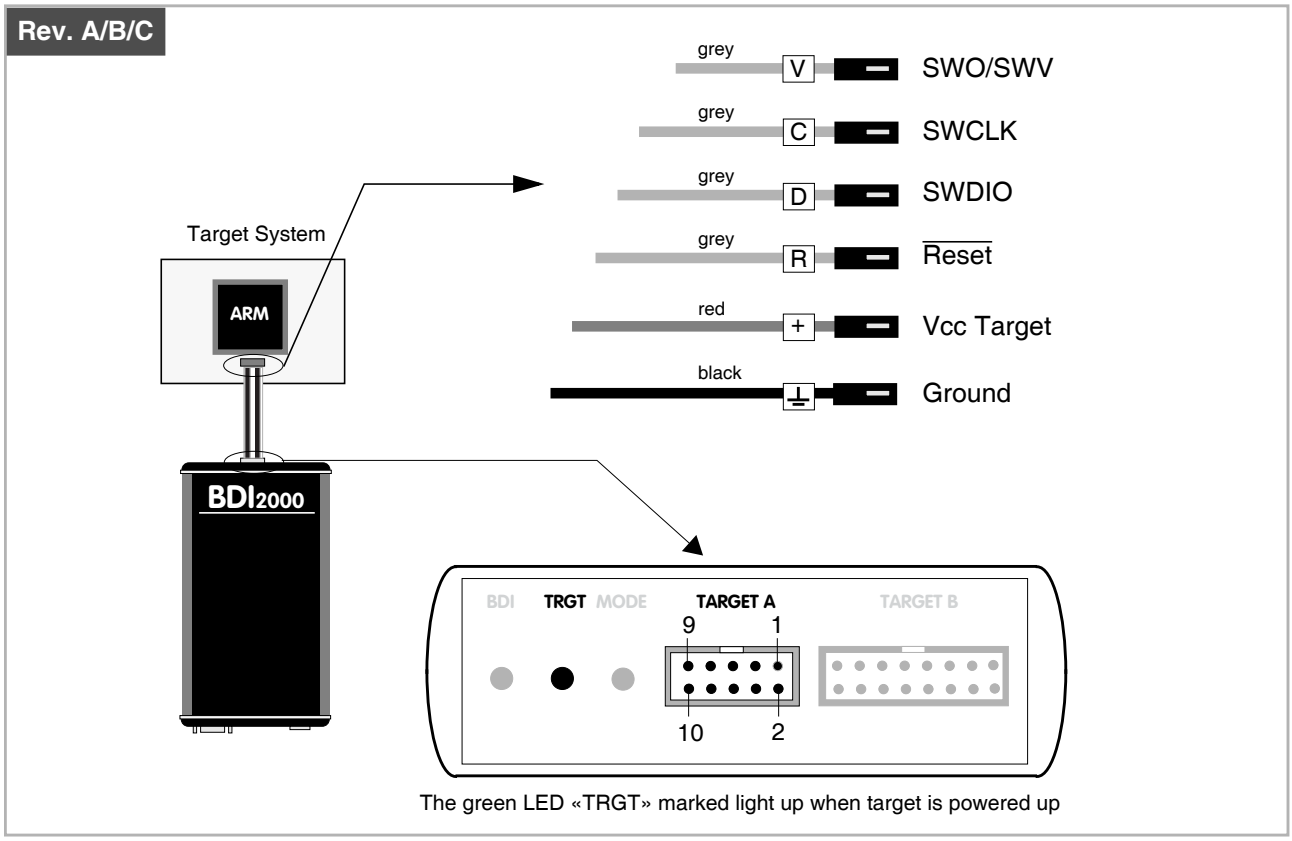
BDI TARGET B Connector Signals:

Pin	Name	Describtion
1	TDO	**JTAG Test Data Out** This input to the BDI2000 connects to the target TDO line.
2	reserved	
3	TDI	**JTAG Test Data In** This output of the BDI2000 connects to the target TDI line.
4	reserved	
5	RTCK	**Returned JTAG Test Clock** This input to the BDI2000 connects to the target RTCK line.
6	Vcc Target	**1.8 – 5.0V:** This is the target reference voltage. It indicates that the target has power and it is also used to create the logic-level reference for the input comparators. It also controls the output logic levels to the target. It is normally fed from Vdd I/O on the target board. **3.0 – 5.0V with Rev. A/B :** This input to the BDI2000 is used to detect if the target is powered up. If there is a current limiting resistor between this pin and the target Vdd, it should be 100 Ohm or less.
7	TCK	**JTAG Test Clock** This output of the BDI2000 connects to the target TCK line.
8	$\overline{\text{TRST}}$	**JTAG Test Reset** This open-drain / push-pull output of the BDI2000 resets the JTAG TAP controller on the target. Default driver type is open-drain.
9	TMS	**JTAG Test Mode Select** This output of the BDI2000 connects to the target TMS line.
10	reserved	
11	reserved	
12	GROUND	**System Ground**
13	$\overline{\text{RESET}}$	**System Reset** This open-drain output of the BDI2000 is used to reset the target system.
14	reseved	
15	reseved	
16	GROUND	**System Ground**

© Copyright 1997-2015 by ABATRON AG Switzerland

2.1.1 Serial Wire Debug

The BDI2000 supports also the „Serial Wire Debug Port“ (SW-DP). In order to use SW-DP a different firmware/logic has to be loaded into the BDI2000 (included on the CD). Also a special target cable is available on request (p/n 90054).



The green LED «TRGT» marked light up when target is powered up

BDI MAIN / TARGET A Connector Signals

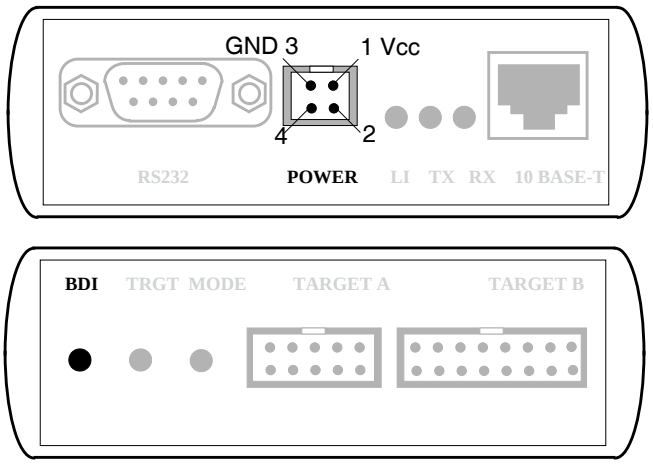
Pin	Name	Describtion
3	GND	System Ground
4	SWCLK	Serial Wire Clock
6	SWDIO	Serial Wire Debug Data Input/Output
10	SWO/SWV	Serial Wire Output / Viewer (optional trace data output)
7	$\overline{\text{RESET}}$	This open collector output of the BDI2000 can be used to hard reset the target system.
9	Vcc Target	**1.8 – 5.0V:** This is the target reference voltage. It indicates that the target has power and it is also used to create the logic-level reference for the input comparators. It also controls the output logic levels to the target. It is normally fed from Vdd I/O on the target board. **3.0 – 5.0V with Rev. A/B :** This input to the BDI2000 is used to detect if the target is powered up. If there is a current limiting resistor between this pin and the target Vdd, it should be 100 Ohm or less.

© Copyright 1997-2015 by ABATRON AG Switzerland

2.2 Connecting the BDI2000 to Power Supply

The BDI2000 needs to be supplied with 5 Volts (max. 1A) via the POWER connector. The available power supply from Abatron (option) or the enclosed power cable can be directly connected. In order to ensure reliable operation of the BDI2000, keep the power supply cable as short as possible.

For error-free operation, the power supply to the BDI2000 must be between 4.75V and 5.25V DC. **The maximal tolerable supply voltage is 5.25 VDC. Any higher voltage or a wrong polarity might destroy the electronics.**



POWER Connector

1 - Vcc (+5V)
2 - VccTGT
3 - GROUND
4 - NOT USED

The green LED «BDI» marked light up when 5V power is connected to the BDI2000

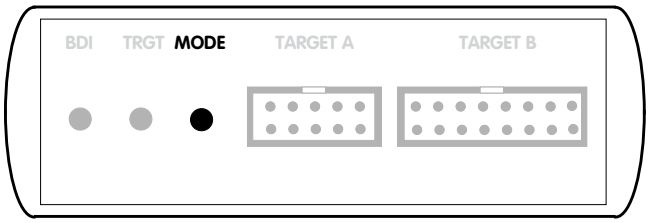
Please switch on the system in the following sequence:

- 1 --> external power supply
- 2 --> target system

© Copyright 1997-2015 by ABATRON AG Switzerland

2.3 Status LED «MODE»

The built in LED indicates the following BDI states:



MODE LED	BDI STATES
OFF	The BDI is ready for use, the firmware is already loaded.
ON	The power supply for the BDI2000 is < 4.75VDC.
BLINK	The BDI «loader mode» is active (an invalid firmware is loaded or loading firmware is active).

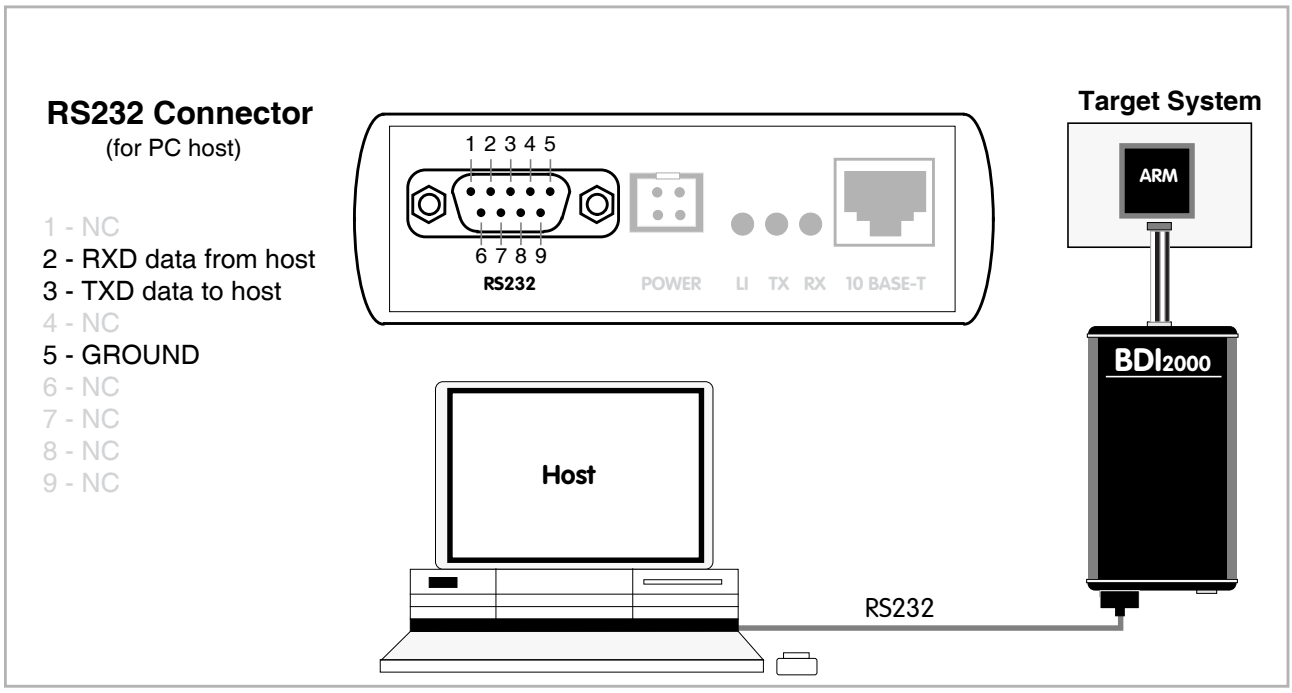
© Copyright 1997-2015 by ABATRON AG Switzerland

2.4 Connecting the BDI2000 to Host

2.4.1 Serial line communication

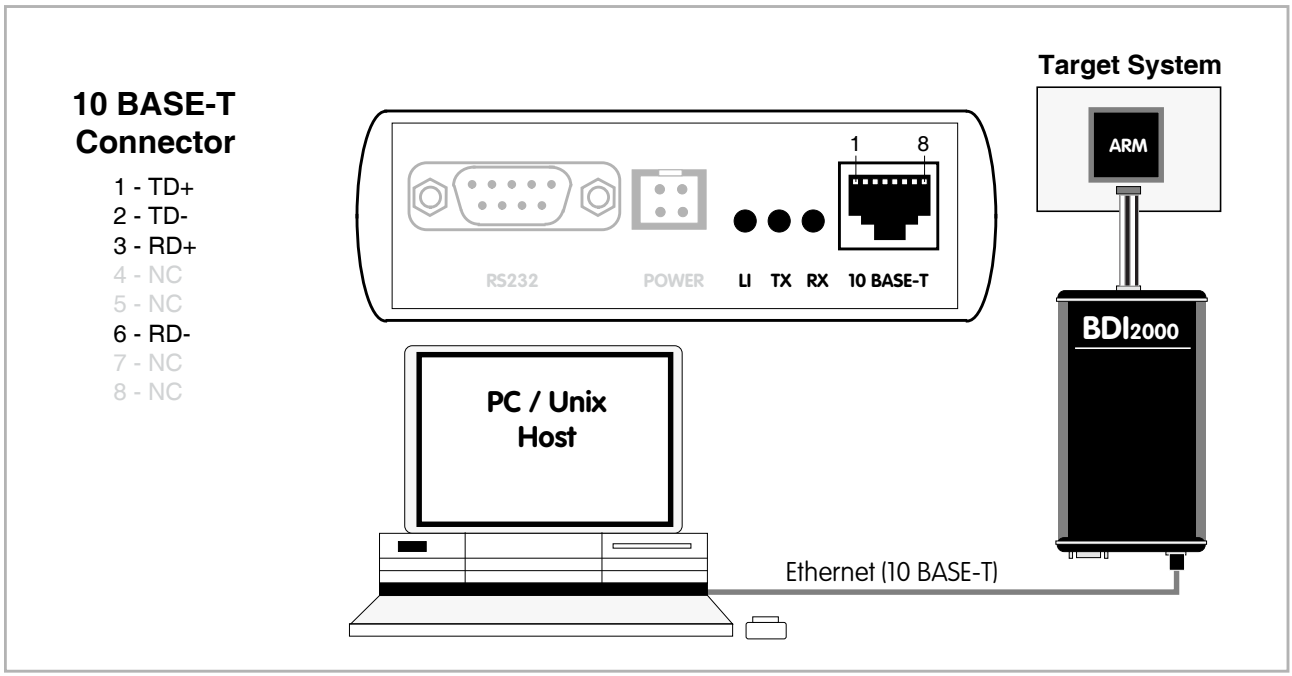
Serial line communication is only used for the initial configuration of the bdiGDB system.

The host is connected to the BDI through the serial interface (COM1...COM4). The communication cable (included) between BDI and Host is a serial cable. There is the same connector pinout for the BDI and for the Host side (Refer to Figure below).



2.4.2 Ethernet communication

The BDI2000 has a built-in 10 BASE-T Ethernet interface (see figure below). Connect an UTP (Unshilded Twisted Pair) cable to the BD2000. For thin Ethernet coaxial networks you can connect a commercially available media converter (BNC-->10 BASE-T) between your network and the BDI2000. Contact your network administrator if you have questions about the network.



The following explains the meanings of the built-in LED lights:

LED	Name	Description
LI	Link	When this LED light is ON, data link is successful between the UTP port of the BDI2000 and the hub to which it is connected.
TX	Transmit	When this LED light BLINKS, data is being transmitted through the UTP port of the BDI2000
RX	Receive	When this LED light BLINKS, data is being received through the UTP port of the BDI2000

© Copyright 1997-2015 by ABATRON AG Switzerland

2.5 Installation of the Configuration Software

On the enclosed CD you will find the BDI configuration software and the firmware required for the BDI2000. For Windows users there is also a simple TFTP server included.

The following files are on the CD.

gdbav821.zip	ZIP achive with the JTAG Mode firmware
gdbsv821.zip	ZIP archive with the Serial Wire Mode firmware

The following files are in the ZIP archives.

b20av8gd.exe / b20sv8gd.exe	Windows configuration program
b20av8gd.xxx / b20sv8gd.xxx	Firmware for the BDI2000
armjed20.xxx / swdjed20.xxx	JEDEC file for the BDI2000 (Rev. A/B) logic device
armjed21.xxx / swdjed21.xxx	JEDEC file for the BDI2000 (Rev. C) logic device
tftpsrv.exe	TFTP server for Windows (WIN32 console application)
*.cfg	Configuration files
*.def	Register definition files
bdisetup.zip	ZIP Archive with the Setup Tool sources for Linux / UNIX hosts.

Overview of an installation / configuration process:

- Create a new directory on your hard disk
- Copy the entire contents of the enclosed CD into this directory
- Linux only: extract the setup tool sources and build the setup tool
- Use the setup tool to load/update the BDI firmware/logic
  Note: A new BDI has no firmware/logic loaded.
- Use the setup tool to transmit the initial configuration parameters
  - IP address of the BDI.
  - IP address of the host with the configuration file.
  - Name of the configuration file. This file is accessed via TFTP.
  - Optional network parameters (subnet mask, default gateway).

Activating BOOTP:
The BDI can get the network configuration and the name of the configuration file also via BOOTP. For this simple enter 0.0.0.0 as the BDI's IP address (see following chapters). If present, the subnet mask and the default gateway (router) is taken from the BOOTP vendor-specific field as defined in RFC 1533.

With the Linux setup tool, simply use the default parameters for the -c option:

```
[root@LINUX_1 bdisetup]# ./bdisetup -c -p/dev/ttyS0 -b57
```

The MAC address is derived from the serial number as follows:
MAC: 00-0C-01-xx-xx-xx , repace the xx-xx-xx with the 6 left digits of the serial number
Example: SN# 93123457 ==>> 00-0C-01-93-12-34

© Copyright 1997-2015 by ABATRON AG Switzerland

2.5.1 Configuration with a Linux / Unix host

The firmware / logic update and the initial configuration of the BDI2000 is done with a command line utility. In the ZIP Archive bdisetup.zip are all sources to build this utility. More information about this utility can be found at the top in the bdisetup.c source file. There is also a make file included.
Starting the tool without any parameter displays information about the syntax and parameters.

To avoid data line conflicts, the BDI2000 must be disconnected from the target system while programming the logic for an other target CPU (see Chapter 2.1.1).

Following the steps to bring-up a new BDI2000:

1. Build the setup tool:
The setup tool is delivered only as source files. This allows to build the tool on any Linux / Unix host. To build the tool, simply start the make utility.

```
[root@LINUX_1 bdisetup]# make
cc -O2   -c -o bdisetup.o bdisetup.c
cc -O2   -c -o bdicnf.o bdicnf.c
cc -O2   -c -o bdidll.o bdidll.c
cc -s bdisetup.o bdicnf.o bdidll.o -o bdisetup
```

2. Check the serial connection to the BDI:
With "bdisetup -v" you may check the serial connection to the BDI. The BDI will respond with information about the current loaded firmware and network configuration.
Note: Login as root, otherwise you probably have no access to the serial port.

```
[root@LINUX_1 bdisetup]# ./bdisetup -v -p/dev/ttyS0 -b57
BDI Type : BDI2000 Rev.C (SN: 92152150)
Loader   : V1.05
Firmware : unknown
Logic    : unknown
MAC      : 00-0c-01-92-15-21
IP Addr  : 255.255.255.255
Subnet   : 255.255.255.255
Gateway  : 255.255.255.255
Host IP  : 255.255.255.255
Config   : ????????????????????
```

3. Load/Update the BDI firmware/logic:
With "bdisetup -u" the firmware is loaded and the CPLD within the BDI2000 is programmed. This configures the BDI for the target you are using. Based on the parameters -a and -t, the tool selects the correct firmware / logic files. If the firmware / logic files are in the same directory as the setup tool, there is no need to enter a -d parameter.

```
[root@LINUX_1 bdisetup]# ./bdisetup -u -p/dev/ttyS0 -b57 -aGDB -tARMV8
Connecting to BDI loader
Erasing CPLD
Programming firmware with ./b20av8gd.100
Programming CPLD with ./armjed21.102
```

Note: for Serial Wire Mode use -tARMSV8 instead of -tARMV8

© Copyright 1997-2015 by ABATRON AG Switzerland

4. Transmit the initial configuration parameters:
With "bdisetup -c" the configuration parameters are written to the flash memory within the BDI.
The following parameters are used to configure the BDI:

BDI IP Address	The IP address for the BDI2000. Ask your network administrator for assigning an IP address to this BDI2000. Every BDI2000 in your network needs a different IP address.
Subnet Mask	The subnet mask of the network where the BDI is connected to. A subnet mask of 255.255.255.255 disables the gateway feature. Ask your network administrator for the correct subnet mask. If the BDI and the host are in the same subnet, it is not necessary to enter a subnet mask.
Default Gateway	Enter the IP address of the default gateway. Ask your network administrator for the correct gateway IP address. If the gateway feature is disabled, you may enter 255.255.255.255 or any other value.
Config - Host IP Address	Enter the IP address of the host with the configuration file. The configuration file is automatically read by the BDI after every start-up via TFTP. If the host IP is 255.255.255.255 then the setup tool stores the configuration read from the file into the BDI internal flash memory. In this case no TFTP server is necessary.
Configuration file	Enter the full path and name of the configuration file. This file is read by the setup tool or via TFTP. Keep in mind that TFTP has it's own root directory (usual /tftpboot).

```
[root@LINUX_1 bdisetup]# ./bdisetup -c -p/dev/ttyS0 -b57 \
> -i151.120.25.101 \
> -h151.120.25.118 \
> -feval7t.cnf
Connecting to BDI loader
Writing network configuration
Writing init list and mode
Configuration passed
```

5. Check configuration and exit loader mode:
The BDI is in loader mode when there is no valid firmware loaded or you connect to it with the setup tool. While in loader mode, the Mode LED is flashing. The BDI will not respond to network requests while in loader mode. To exit loader mode, the "bdisetup -v -s" can be used. You may also power-off the BDI, wait some time (1min.) and power-on it again to exit loader mode.

```
[root@LINUX_1 bdisetup]# ./bdisetup -v -p/dev/ttyS0 -b57 -s
BDI Type : BDI2000 Rev.C (SN: 92152150)
Loader   : V1.05
Firmware : V1.00 bdiGDB for ARMV8
Logic    : V1.02 ARM
MAC      : 00-0c-01-92-15-21
IP Addr  : 151.120.25.101
Subnet   : 255.255.255.255
Gateway  : 255.255.255.255
Host IP  : 151.120.25.118
Config   : eval7t.cnf
```

The Mode LED should go off, and you can try to connect to the BDI via Telnet.

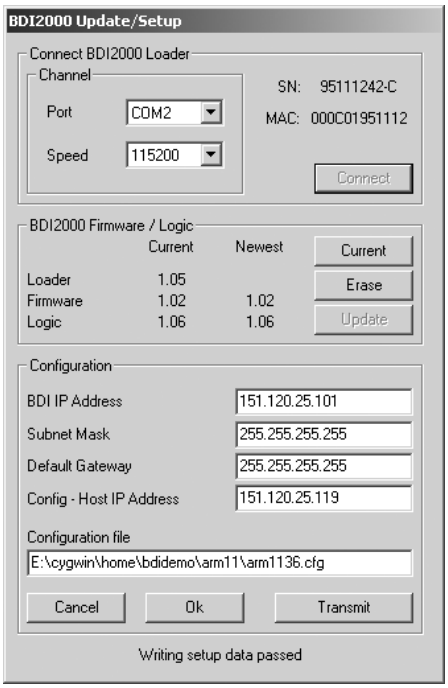
```
[root@LINUX_1 bdisetup]# telnet 151.120.25.101
```

© Copyright 1997-2015 by ABATRON AG Switzerland

2.5.2 Configuration with a Windows host

First make sure that the BDI is properly connected (see Chapter 2.1 to 2.4).

To avoid data line conflicts, the BDI2000 must be disconnected from the target system while programming the logic for an other target CPU (see Chapter 2.1.1).



dialog box «BDI2000 Update/Setup»

Before you can use the BDI2000 together with the GNU debugger, you must store the initial configuration parameters in the BDI2000 flash memory. The following options allow you to do this:

Channel	Select the communication port where the BDI2000 is connected during this setup session.
Baudrate	Select the baudrate used to communicate with the BDI2000 loader during this setup session.
Connect	Click on this button to establish a connection with the BDI2000 loader. Once connected, the BDI2000 remains in loader mode until it is restarted or this dialog box is closed.
Current	Press this button to read back the current loaded BDI2000 software and logic versions. The current loader, firmware and logic version will be displayed.
Update	This button is only active if there is a newer firmware or logic version present in the execution directory of the bdiGDB setup software. Press this button to write the new firmware and/or logic into the BDI2000 flash memory / programmable logic.

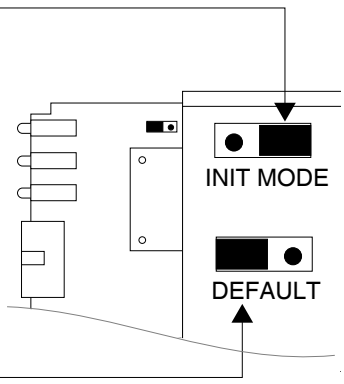
© Copyright 1997-2015 by ABATRON AG Switzerland

BDI IP Address	Enter the IP address for the BDI2000. Use the following format: xxx.xxx.xxx.xxx e.g.151.120.25.101 Ask your network administrator for assigning an IP address to this BDI2000. Every BDI2000 in your network needs a different IP address.
Subnet Mask	Enter the subnet mask of the network where the BDI is connected to. Use the following format: xxx.xxx.xxx.xxxe.g.255.255.255.0 A subnet mask of 255.255.255.255 disables the gateway feature. Ask your network administrator for the correct subnet mask.
Default Gateway	Enter the IP address of the default gateway. Ask your network administrator for the correct gateway IP address. If the gateway feature is disabled, you may enter 255.255.255.255 or any other value..
Config - Host IP Address	Enter the IP address of the host with the configuration file. The configuration file is automatically read by the BDI after every start-up via TFTP. If the host IP is 255.255.255.255 then the setup tool stores the configuration read from the file into the BDI internal flash memory. In this case no TFTP server is necessary.
Configuration file	Enter the full path and name of the configuration file. This file is read by the setup tool or via TFTP.
Transmit	Click on this button to store the configuration in the BDI2000 flash memory.

2.5.3 Recover procedure

In rare instances you may not be able to load the firmware in spite of a correctly connected BDI (error of the previous firmware in the flash memory). **Before carrying out the following procedure, check the possibilities in Appendix «Troubleshooting»**. In case you do not have any success with the tips there, do the following:

- Switch OFF the power supply for the BDI and open the unit as described in Appendix «Maintenance»
- Place the jumper in the **«INIT MODE»** position
- Connect the power cable or target cable if the BDI is powered from target system
- Switch ON the power supply for the BDI again and wait until the LED «MODE» blinks fast
- Turn the power supply OFF again
- Return the jumper to the **«DEFAULT»** position
- Reassemble the unit as described in Appendix «Maintenance»



© Copyright 1997-2015 by ABATRON AG Switzerland

2.6 Testing the BDI2000 to host connection

After the initial setup is done, you can test the communication between the host and the BDI2000. There is no need for a target configuration file and no TFTP server is needed on the host.

- If not already done, connect the bdiGDB system to the network.
- Power-up the BDI2000.
- Start a Telnet client on the host and connect to the BDI2000 (the IP address you entered during initial configuration).
- If everything is okay, a sign on message like «BDI Debugger for ARM» should be displayed in the Telnet window.

2.7 TFTP server for Windows

The bdiGDB system uses TFTP to access the configuration file and to load the application program. Because there is no TFTP server bundled with Windows NT, Abatron provides a TFTP server application **tftpsrv.exe**. This WIN32 console application runs as normal user application (not as a system service).

Command line syntax: tftpsrv [p] [w] [dRootDirectory]

Without any parameter, the server starts in read-only mode. This means, only read access request from the client are granted. This is the normal working mode. The bdiGDB system needs only read access to the configuration and program files.

The parameter [p] enables protocol output to the console window. Try it.
The parameter [w] enables write accesses to the host file system.
The parameter [d] allows to define a root directory.

`tftpsrv p`	Starts the TFTP server and enables protocol output
`tftpsrv p w`	Starts the TFTP server, enables protocol output and write accesses are allowed.
`tftpsrv dC:\tftp\`	Starts the TFTP server and allows only access to files in C:\tftp and its subdirectories. As file name, use relative names. For example "bdi\mpc750.cfg" accesses "C:\tftp\bdi\mpc750.cfg"

You may enter the TFTP server into the Startup group so the server is started every time you logon.

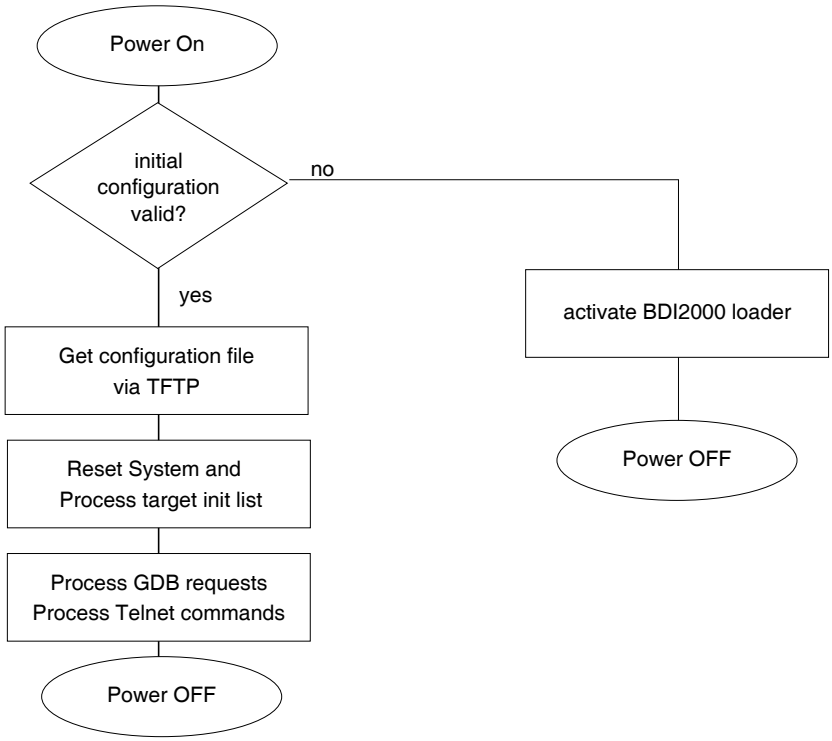
© Copyright 1997-2015 by ABATRON AG Switzerland

3 Using bdiGDB

3.1 Principle of operation

The firmware within the BDI handles the GDB request and accesses the target memory or registers via the JTAG interface. There is no need for any debug software on the target system. After loading the code via TFTP debugging can begin at the very first assembler statement.

Whenever the BDI system is powered-up the following sequence starts:



© Copyright 1997-2015 by ABATRON AG Switzerland

3.2 Configuration File

The configuration file is automatically read by the BDI after every power on.
The syntax of this file is as follows:

```
; comment
[part name]
core# identifier parameter1 parameter2 ..... parameterN  ; comment
core# identifier parameter1 parameter2 ..... parameterN
.....
[part name]
core# identifier parameter1 parameter2 ..... parameterN
core# identifier parameter1 parameter2 ..... parameterN
.....
             etc.
```

Numeric parameters can be entered as decimal (e.g. 700) or as hexadecimal (0x80000).

The core# is optional. If not present the BDI assume core #0. See also chapter "Multi-Core Support".

Note about how to enter 64bit values:

The syntax for 64 bit parameters is : [<high word>_]<low word>
Hex values may also be entered as: 0xnnnnnnnnnnnnnnnn

The "high word" (optional) and "low word" can be entered as decimal or hexadecimal. They are handled as two separate values concatenated with an underscore.

Examples:

```
0x0123456789abcdef         =>>    0x0123456789abcdef
0x01234567_0x89abcdef      =>>    0x0123456789abcdef
1_0                        =>>    0x0000000100000000
256                        =>>    0x0000000000000100
3_0x1234                   =>>    0x0000000300001234
0x80000000_0               =>>    0x8000000000000000
```

© Copyright 1997-2015 by ABATRON AG Switzerland

3.2.1 Part [INIT]

The part [INIT] defines a list of commands which are be executed every time the target comes out of reset (except in STARTUP RUN mode). The commands are used to get the processor ready.

WGPR register value — Write value to the selected general purpose register.
- register — the register number 0 .. 31
- value — the value to write into the register

Example: WGPR 0 5

WREG name value — Write value to the selected register/memory by name
- name — the case sensitive register name from the reg def file
- value — the value to write to the register/memory

Example: WREG sp 0x1d00fff0

WSYS register value — Write value to the selected system register.
- register — the register number (see chapter Telnet interface)
- value — the value to write into the register
- Example: WSYS 0x8200 ; set EL1 Translation Base 0

WM8 address value — Write a byte (8bit) to the selected memory place.
- address — the memory address
- value — the value to write to the target memory

Example: WM8 0x1d000000 0x01

WM16 address value — Write a half word (16bit) to the selected memory place.
- address — the memory address
- value — the value to write to the target memory

Example: WM16 0x1d000000 0x0002

WM32 address value — Write a word (32bit) to the selected memory place.
- address — the memory address
- value — the value to write to the target memory

Example: WM32 0x1d000000 0x12345678

WAPB address value — Write a word (32bit) to the APB memory.
- address — the APB memory address
- value — the value to write to the APB memory

Example: WAPB 0xfc010400 0x1d000100 ; DBGBVR0_EL1

WDBG offset value — Write a word (32bit) to an external debug register.
- offset — the offset to the external debug register
- value — the value to write to the register

Example: WDBG 0x400 0x1d000100 ; DBGBVR0_EL1

© Copyright 1997-2015 by ABATRON AG Switzerland

RM8 address [xor] Read a byte (8bit) from the selected memory place.

RM16 address [xor] Read a half word (16bit) from the selected memory place.

RM32 address [xor] Read a word (32bit) from the selected memory place.

address	the memory address
xor	optional XOR pattern applied to the read value
Example:	RM32 0x00000000

WMX and or Writes back a modified read value. The address and size is the same as used by RM8, RM16 or RM32. This allows simple bit manipulations.

and	the AND pattern applied to the read value
or	the OR pattern applied to the read value
Example:	RM32 0x200000000 0x10101010 ; read and XOR WMX 0xff00ff00 0x00000003 ; AND, OR and write back

WAIT mask equal Waits until ((memory & mask) == equal). The last RM8, RM16 or RM32 entry defines the address and the size for the following WAIT.

mask	the bit mask used before comparing
equal	the value to compare against
Example:	RM16 0x2000000a WAIT 0x000f0ff 0x00001034 ; wait until equal

MMAP start end Because a memory access to an invalid memory space via JTAG leads to a deadlock, this entry can be used to define up to 32 valid memory ranges. If at least one memory range is defined, the BDI checks against this range(s) and avoids accessing of not mapped memory ranges.

start	the start address of a valid memory range
end	the end address of this memory range
Example:	MMAP 0xFFE00000 0xFFFFFFFF ;Boot ROM

DELAY value Delay for the selected time.

value	the delay time in milliseconds (1...30000)
Example:	DELAY 500 ; delay for 0.5 seconds

CLOCK value This entry allows to change the JTAG clock frequency during processing of the init list. But the final JTAG clock after processing the init list is taken from the CLOCK entry in the [TARGET] section. This entry maybe of interest to speed-up JTAG clock as soon as possible (after PLL setup).

value	see CLOCK parameter in [TARGET] section
Example:	CLOCK 2 ; switch to 8 MHz JTAG clock

EXEC [n_]opcode [data] This entry causes the processor to execute one instruction. The data is loaded to GPRn before the instruction is executed.

n	the GPR number (default is 0)
opcode	the opcode of the instruction
data	the data loaded to GPRn (default is 0)
Example:	EXEC 0xd508751f ;IC invalidate all

© Copyright 1997-2015 by ABATRON AG Switzerland

3.2.2 Part [TARGET]

The part [TARGET] defines some target specific values.

CPUTYPE type [{ index | addr }]

This value gives the BDI information about the connected CPU.

type	The CPU type from the following list: CORTEX-A50, X-GENE, THUNDERX
index	Defines which core debug component to select(0..7).
addr	Specifies the APB address of the core debug component. There is no ROM table search in this case.
Example:	CPUTYPE X-GENE 0xfc010000 ;X-Gene CPU 0

CTI addr [cgroup]

This entry allows to override the default base address of the CTI component and optionally to define the core group parameter.

addr	Defines the APB address of the Cross-Trigger Interface (CTI) component. Default is debug base + 0x10000.
cgroup	This is a bitmap of selected cores. It gives the BDI information about how to restart multiple cores in response to a GDB continue command. See chapter Multi-Core Support.
Example:	#0 CTI 0xfc020000 0xff ;CTI base, core group master #1 CTI 0xfc120000 0x02 ;CTI base, core group slave #2 CTI 0xfc220000 0x04 ;CTI base, core group slave

CLOCK main [init] [SLOW]

With this value(s) you can select the JTAG clock rate the BDI2000 uses when communicating with the target processor. The "main" entry is used after processing the initialization list. The "init" value is used after target reset until the initialization list is processed. If there is no "init" value defined, the "main" value is used all the times.
Adaptive clocking is only supported with BDI2000 Rev.B/C and needs a special target connector cable. Add also SLOW if the CPU clock frequency may fall below 6 MHz during adaptive clocking.

main,init:	The clock frequency in Hertz or an index value from the following table: 0 = Adaptive 1 = 16 MHz 6 = 200 kHz 2 = 8 MHz 7 = 100 kHz 3 = 4 MHz 8 = 50 kHz 4 = 1 MHz 9 = 20 kHz 5 = 500 kHz 10 = 10 kHz
Example:	CLOCK 2 ; 16 MHz JTAG clock CLOCK 8000000 ; 8 MHz JTAG clock

© Copyright 1997-2015 by ABATRON AG Switzerland

TRST type

Normally the BDI uses an open drain driver for the TRST signal. This is in accordance with the ARM recommendation. For boards where TRST is simply pulled low with a weak resistor, TRST will always be asserted and JTAG debugging is impossible. In that case, the TRST driver type can be changed to push-pull. Then the BDI actively drives also high level.

type	OPENDRAIN (default) PUSHPULL
Example:	TRST PUSHPULL ; Drive TRST also high

RESET type [time] [pwr]

Normally the BDI drives the reset line during a reset sequence. If reset type is NONE or SOFT, the BDI does not assert a hardware reset. If reset type SOFT is supported depends on the connected target.

type	NONE (default) SOFT (soft reset via a debug register) HARD
time	The time in milliseconds the BDI assert the reset signal.
pwr	A different reset type can be defined for the initial power-up reset (NONE, SOFT, HARD).
Example:	RESET SOFT ; reset ARM core via RCSR RESET HARD 1000 ; assert RESET for 1 second

STARTUP mode [runtime]

This parameter selects the target startup mode. The following modes are supported:

HALT	This default mode tries to forces the target to debug mode immediately out of reset.
STOP	In this mode, the BDI lets the target execute code for "runtime" milliseconds after reset. This mode is useful when boot code should initialize the target system.
RUN	After reset, the target executes code until stopped by the Telnet "halt" command. The init list is not processed in this mode.
WAIT	Sets the debug request bit in the target. Once the target is released from reset it will enter debug mode.
IDLE	In this mode, the BDI does not access the target/core until it is attached via the Telnet „attach“ command. This is useful for cores that are not accessible after reset. Only after the attach, the BDI starts communicating with the debug logic of this target/core.
Example:	STARTUP STOP 3000 ; let the CPU run for 3 seconds

WAKEUP time

This entry in the init list allows to define a delay time (in ms) the BDI inserts between releasing the reset line and starting communicating with the target. This delay is necessary when a target needs some wake-up time after a reset.

time	the delay time in milliseconds
Example:	WAKEUP 3000 ; insert 3sec wake-up time

© Copyright 1997-2015 by ABATRON AG Switzerland

BDIMODE mode param

This parameter selects the BDI debugging mode. The following modes are supported:

LOADONLY	Loads and starts the application code. No debugging via JTAG interface.
AGENT	The debug agent runs within the BDI (default mode). This mode accepts a second parameter. If RUN is entered as a second parameter, the loaded application will be started immediately, otherwise only the PC is set and BDI waits for GDB requests. If QUIET is entered as a second parameter, the BDI no longer polls the debug status register. The target is not influenced in any way while it is running. But in this mode, the BDI cannot detect any debug mode entry.
Example:	BDIMODE AGENT RUN

ENDIAN format

This entry defines the endiannes of the memory system.

format	The endiannes of the target memory: LITTLE (default) BIG
Example:	ENDIAN LITTLE

VECTOR CATCH list

When this line is present, the BDI catches some events. Reset catch, OS unlock catch and SW access to debug register trap can individually enabled. An empty list enables all of them.

list	List of events to catch: RST Reset catch OSU OS unlock catch TDA Trap access to debug registers
Example:	VECTOR CATCH ;catch all events VECTOR CATCH RST ;catch only reset event

BREAKMODE mode

This parameter defines how the BDI handles normal breakpoints set via GDB "break" command. The Telnet "bi" command always sets a hardware breakpoint.

SOFT	This is the default mode. Breakpoints are set by replacing code with a HLT instruction.
HARD	In this mode, the breakpoint hardware is used. The number of available breakpoints depends on the target.
Example:	BREAKMODE HARD

© Copyright 1997-2015 by ABATRON AG Switzerland

MEMACCES mode [wait [hprot]]

This parameter defines how memory is accessed. Either via the ARM core by executing "ld" and "st" instructions or via the AHB/AXI access port. The current mode can also be changed via the Telnet interface. The optional wait parameter allows to define a time the BDI waits before it expects that a value is ready or written. This allows to optimize download performance. The wait time is (8 x wait) TCK's in Run-Test/Idle state.
The hprot option allows to define the CSW[31:24] bits in the AHB/AXI-AP.
The following modes are supported:

CORE	The CORE (default) mode requires that the core is halted and makes use of the memory management unit (MMU) and cache.
AHB or AXI	The AHB/AXI access mode can access memory even when the core is running but bypasses MMU and cache. **Note:** The BDI automatically handles the AHB to AXI mapping for X-Gene.
Example:	MEMACCES CORE 5 ; 40 TCK's access delay MEMACCES AHB 4 ; access via AHB, 32 TCK delay

SIO port [baudrate]

When this line is present, a TCP/IP channel is routed to the BDI's RS232 connector. The port parameter defines the TCP port used for this BDI to host communication. You may choose any port except 0 and the default Telnet port (23). On the host, open a Telnet session using this port. Now you should see the UART output in this Telnet session. You can use the normal Telnet connection to the BDI in parallel, they work completely independent. Also input to the UART is implemented.

port	The TCP/IP port used for the host communication.
baudrate	The BDI supports 2400 ... 115200 baud
Example:	SIO 7 9600 ;TCP port for virtual IO

DCC port

When this line is present, a TCP/IP channel is routed to the ARM debug communication channel (DCC). The port parameter defines the TCP port used for this BDI to host communication. You may choose any port except 0 and the default Telnet port (23). On the host, open a Telnet session using this port. Now you should see the DCC output in this Telnet session. You can use the normal Telnet connection to the BDI in parallel, they work completely independent. Also input to DCC is implemented.

port	The TCP/IP port used for the host communication.
Example:	DCC 7 ;TCP port for DCC I/O

© Copyright 1997-2015 by ABATRON AG Switzerland

SWO port baudrate

Only supported in Serial Wire Mode!
When this line is present, a TCP/IP channel is routed to the Serial Wire Output (SWO/SWV). The port parameter defines the TCP port used for this BDI to host communication. You may choose any port except 0 and the default Telnet port (23). If an even port number is used (raw mode), the BDI sends all data received via SWO in hexadecimal format to the host. For an odd port number (ASCII mode), the bytes received in the range 4 to 127 are directly forwared to the host, all other bytes are discarded. On the host, open a Telnet session using this port. Now you should see the Serial Wire Output in this Telnet session.

port	The TCP/IP port used for the host communication.
baudrate	The BDI2000 supports 2400 ... 115200 baud and 122kb, 130kb, 139kb, 149kb, 160kb, 174kb, 189kb, 208kb, 232kb, 260kb, 298kb, 347kb, 417kb, 520kb
Example:	SWO 8023 260000 ;map ASCII SWO to odd port 8023 SWO 8020 260000 ;map raw SWO to even port 8020

Daisy chained JTAG devices:
For ARM targets, the BDI can also handle systems with multiple devices connected to the JTAG scan chain. In order to put the other devices into BYPASS mode and to count for the additional bypass registers, the BDI needs some information about the scan chain layout. Enter the number (count) and total instruction register (irlen) length of the devices present before the ARM chip (Predecessor). Enter the appropriate information also for the devices following the ARM chip (Successor):

SCANPRED count irlen

This value gives the BDI information about JTAG devices present before the ARM chip in the JTAG scan chain.

count	The number of preceding devices
irlen	The sum of the length of all preceding instruction registers (IR).
Example:	SCANPRED 1 8 ; one device with an IR length of 8

SCANSUCC count irlen

This value gives the BDI information about JTAG devices present after the ARM chip in the JTAG scan chain.

count	The number of succeeding devices
irlen	The sum of the length of all succeeding instruction registers (IR).
Example:	SCANSUCC 2 12 ; two device with an IR length of 8+4

Note:
For Serial Wire Mode, the following parameters are not relevant, have no function:
TRST, SCANPRED, SCANSUCC

© Copyright 1997-2015 by ABATRON AG Switzerland

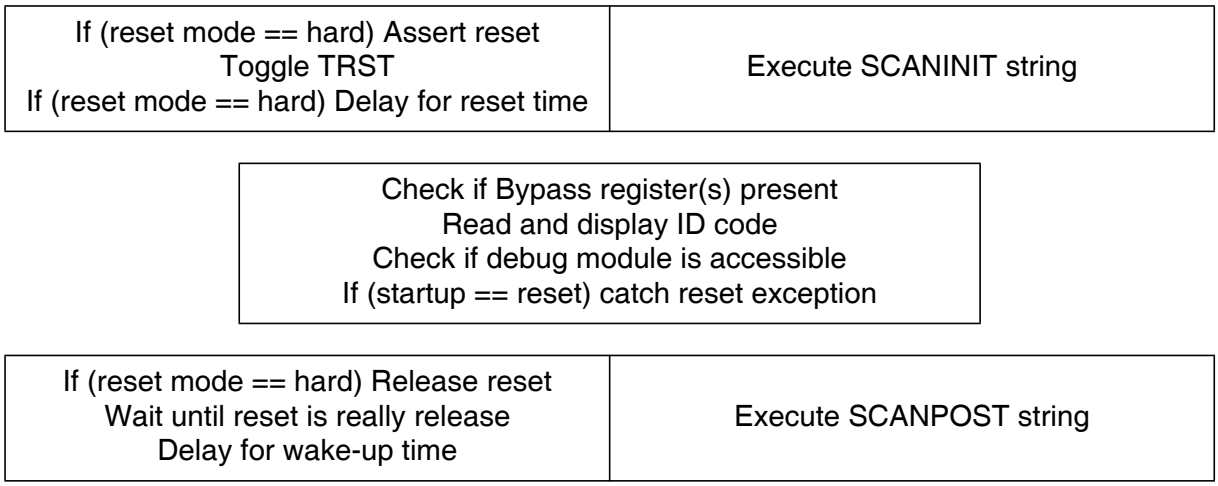
Low level JTAG scan chain configuration:
Sometimes it is necessary to configure the test access port (TAP) of the target before the ARM debug interface is visible and accessible in the usual way. The BDI supports this configuration in a very generic way via the SCANINIT and SCANPOST configuration commands. Both accept a string that defines the JTAG sequences to execute. The following example shows how to use these commands:

```
; Configure ICEPick module to make ARM926 TAP visible
SCANINIT    t1:w1000:t0:w1000:     ;toggle TRST
SCANINIT    i6=07:d8=89:i6=02:     ;connect and select router
SCANINIT    d32=81000082:          ;set IP control
SCANINIT    d32=a018206f:          ;configure TAP0
SCANINIT    d32=a018216f:cl5:      ;enable TAP0, clock 5 times in RTI
SCANINIT    i10=ffff               ;scan bypass
;
; Between SCANINIT and SCANPOST the ARM ICEBreaker is configured
; and the DBGRQ bit in the ARM debug control register is set.
;
SCANPOST    i10=002f:              ;IP(router) - ARM(bypass)
SCANPOST    d33=0102000106:        ;IP control = SysReset
SCANPOST    i10=ffff               ;scan bypass
```

The following low level JTAG commands are supported in the string. Use ":" between commands.

```
I<n>=<...b2b1b0>   write IR, b0 is first scanned  (not for SWD)
D<n>=<...b2b1b0>   write DR, b0 is first scanned  (not for SWD)
                     n  : the number of bits 1..256
                     bx : a data byte, two hex digits
W<n>               wait for n (decimal) micro seconds
T1                 assert TRST   (not for SWD)
T0                 release TRST  (not for SWD)
R1                 assert RESET
R0                 release RESET
CH<n>              clock TCK n (decimal) times with TMS high  (not for SWD)
CL<n>              clock TCK n (decimal) times with TMS low   (not for SWD)
M<addr>=<data>     write the 32-bit data value to addr in AHB memory space
A<addr>=<data>     write the 32-bit data value to addr in APB memory space
P<addr>=<value>    write 32-bit to Access Port register
```

The following diagram shows the parts of the standard reset sequence that are replaced with the SCAN string. Only the appropriate part of the reset sequence is replaced. If only a SCANINIT string is defined, then the standard "post" sequence is still executed.


If (reset mode == hard) Assert reset
Toggle TRST
If (reset mode == hard) Delay for reset time
Execute SCANINIT string
Check if Bypass register(s) present
Read and display ID code
Check if debug module is accessible
If (startup == reset) catch reset exception
If (reset mode == hard) Release reset
Wait until reset is really release
Delay for wake-up time
Execute SCANPOST string

© Copyright 1997-2015 by ABATRON AG Switzerland

3.2.3 Part [HOST]

The part [HOST] defines some host specific values.

IP ipaddress

The IP address of the host.

ipaddress	the IP address in the form xxx.xxx.xxx.xxx
Example:	IP 151.120.25.100

FILE filename

The default name of the file that is loaded into RAM using the Telnet 'load' command. This name is used to access the file via TFTP. If the filename starts with a $, this $ is replace with the path of the configuration file name.

filename	the filename including the full path or $ for relative path.
Example:	FILE F:\gnu\demo\arm\test.elf FILE $test.elf

FORMAT format [offset]

The format of the image file and an optional load address offset. If the image is already stored in ROM on the target, select ROM as the format. The optional parameter "offset" is added to any load address read from the image file.

format	SREC, BIN, ELF or ROM
Example:	FORMAT ELF FORMAT ELF 0x10000

LOAD mode

In Agent mode, this parameters defines if the code is loaded automatically after every reset.

mode	AUTO, MANUAL
Example:	LOAD MANUAL

START address

The address where to start the program file. If this value is not defined and the core is not in ROM, the address is taken from the code file. If this value is not defined and the core is already in ROM, the PC will not be set before starting the target. This means, the program starts at the normal reset address (0x00000000).

address	the address where to start the program file
Example:	START 0x10000

DEBUGPORT port [RECONNECT]

The TCP port GDB uses to access the target. If the RECONNECT parameter is present, an open TCP/IP connection (Telnet/GDB) will be closed if there is a connect request from the same host (same IP address).

port	the TCP port number (default = 2001)
Example:	DEBUGPORT 2001

© Copyright 1997-2015 by ABATRON AG Switzerland

PROMPT string

This entry defines a new Telnet prompt. The current prompt can also be changed via the Telnet interface.

Example: PROMPT XGENE#0>

DUMP filename

The default file name used for the DUMP command from a Telnet session.

filename the filename including the full path

Example: DUMP dump.bin

TELNET mode

By default the BDI sends echoes for the received characters and supports command history and line editing. If it should not send echoes and let the Telnet client in "line mode", add this entry to the configuration file.

mode ECHO (default), NOECHO or LINE

Example: TELNET NOECHO ; use old line mode

© Copyright 1997-2015 by ABATRON AG Switzerland

3.2.4 Part [FLASH]

The Telnet interface supports programming and erasing of flash memories. The bdiGDB system has to know which type of flash is used, how the chip(s) are connected to the CPU and which sectors to erase in case the ERASE command is entered without any parameter.

CHIPTYPE type

This parameter defines the type of flash used. It is used to select the correct programming algorithm.

format AM29F, AM29BX8, AM29BX16, I28BX8, I28BX16, AT49, AT49X8, AT49X16, STRATAX8, STRATAX16, MIRROR, MIRRORX8, MIRRORX16, S29M32X16, S29GLSX16, S29VSRX16, M58X32, AM29DX16, AM29DX32,

Example: CHIPTYPE AM29F

CHIPSIZE size

The size of **one** flash chip in bytes (e.g. AM29F010 = 0x20000). This value is used to calculate the starting address of the current flash memory bank.

size the size of one flash chip in bytes

Example: CHIPSIZE 0x80000

BUSWIDTH width

Enter the width of the memory bus that leads to the flash chips. Do not enter the width of the flash chip itself. The parameter CHIPTYPE carries the information about the number of data lines connected to one flash chip. For example, enter 16 if you are using two AM29F010 to build a 16bit flash memory bank.

with the width of the flash memory bus in bits (8 | 16 | 32)

Example: BUSWIDTH 16

FILE filename

The default name of the file that is programmed into flash using the Telnet 'prog' command. This name is used to access the file via TFTP. If the filename starts with a $, this $ is replace with the path of the configuration file name. This name may be overridden interactively at the Telnet interface.

filename the filename including the full path or $ for relative path.

Example: FILE F:\gnu\arm\bootrom.hex
FILE $bootrom.hex

FORMAT format [offset]

The format of the file and an optional address offset. The optional parameter "offset" is added to any load address read from the program file.

format SREC, BIN, or ELF

Example: FORMAT SREC
FORMAT ELF 0x10000

WORKSPACE address

If a workspace is defined, the BDI uses a faster programming algorithm that runs out of RAM on the target system. Otherwise, the algorithm is processed within the BDI. The workspace is used for a 1kByte data buffer and to store the algorithm code. There must be at least 2kBytes of RAM available for this purpose.

address the address of the RAM area

Example: WORKSPACE 0x00000000

© Copyright 1997-2015 by ABATRON AG Switzerland

ERASE addr [increment count] [mode [wait]]

The flash memory may be individually erased or unlocked via the Telnet interface. In order to make erasing of multiple flash sectors easier, you can enter an erase list. All entries in the erase list will be processed if you enter ERASE at the Telnet prompt without any parameter. This list is also used if you enter UNLOCK at the Telnet without any parameters. With the "increment" and "count" option you can erase multiple equal sized sectors with one entry in the erase list.

address	Address of the flash sector, block or chip to erase
increment	If present, the address offset to the next flash sector
count	If present, the number of equal sized sectors to erase
mode	BLOCK, CHIP, UNLOCK Without this optional parameter, the BDI executes a sector erase. If supported by the chip, you can also specify a block or chip erase. If UNLOCK is defined, this entry is also part of the unlock list. This unlock list is processed if the Telnet UNLOCK command is entered without any parameters. **Note:** Chip erase does not work for large chips because the BDI time-outs after 3 minutes. Use block erase.
wait	The wait time in ms is only used for the unlock mode. After starting the flash unlock, the BDI waits until it processes the next entry.
Example:	ERASE 0xff040000 ;erase sector 4 of flash ERASE 0xff060000 ;erase sector 6 of flash ERASE 0xff000000 CHIP ;erase whole chip(s) ERASE 0xff010000 UNLOCK 100 ;unlock, wait 100ms ERASE 0xff000000 0x10000 7 ; erase 7 sectors

Example for the ARM PID7T board (AM29F010 in U12):

```
[FLASH]
WORKSPACE   0x00000000  ;Workspace in target RAM for faster programming algorithm
CHIPTYPE    AM29F       ;Flash type
CHIPSIZE    0x20000     ;The size of one flash chip in bytes
BUSWIDTH    8           ;The width of the flash memory bus in bits (8 | 16 | 32)
FILE        C:\gdb\pid7t\bootrom.hex  ;The file to program
ERASE       0x04000000  ;erase sector 0 of flash SIMM
ERASE       0x04004000  ;erase sector 1 of flash SIMM
ERASE       0x04008000  ;erase sector 2 of flash SIMM
ERASE       0x0400C000  ;erase sector 3 of flash SIMM
ERASE       0x04010000  ;erase sector 4 of flash SIMM
ERASE       0x04014000  ;erase sector 5 of flash SIMM
ERASE       0x04018000  ;erase sector 6 of flash SIMM
ERASE       0x0401C000  ;erase sector 7 of flash SIMM
```

the above erase list maybe replaced with:

```
ERASE        0x04000000 0x4000 8 ;erase 8 sectors
```

© Copyright 1997-2015 by ABATRON AG Switzerland

Supported standard parallel NOR Flash Memories:

There are different flash algorithm supported. Almost all currently available parallel NOR flash memories can be programmed with one of these algorithm. The flash type selects the appropriate algorithm and gives additional information about the used flash.

On our web site (www.abatron.ch -> Debugger Support -> GNU Support -> Flash Support) there is a PDF document available that shows the supported parallel NOR flash memories.

Some newer Spansion MirrorBit flashes cannot be programmed with the MIRRORX16 algorithm because of the used unlock address offset. Use S29M32X16 for these flashes.

The AMD and AT49 algorithm are almost the same. The only difference is, that the AT49 algorithm does not check for the AMD status bit 5 (Exceeded Timing Limits).

Only the AMD and AT49 algorithm support chip erase. Block erase is only supported with the AT49 algorithm. If the algorithm does not support the selected mode, sector erase is performed. If the chip does not support the selected mode, erasing will fail. The erase command sequence is different only in the 6th write cycle. Depending on the selected mode, the following data is written in this cycle (see also flash data sheets): 0x10 for chip erase, 0x30 for sector erase, 0x50 for block erase.
To speed up programming of Intel Strata Flash and AMD MirrorBit Flash, an additional algorithm is implemented that makes use of the write buffer. The Strata algorithm needs a workspace, otherwise the standard Intel algorithm is used.

© Copyright 1997-2015 by ABATRON AG Switzerland

Note:
Some Intel flash chips (e.g. 28F800C3, 28F160C3, 28F320C3) power-up with all blocks in locked state. In order to erase/program those flash chips, use the init list to unlock the appropriate blocks:

```
WM16   0xFFF00000   0x0060       unlock block 0
WM16   0xFFF00000   0x00D0
WM16   0xFFF10000   0x0060       unlock block 1
WM16   0xFFF10000   0x00D0
       ....
WM16   0xFFF00000   0xFFFF       select read mode
```

or use the Telnet "unlock" command:

```
UNLOCK [<addr> [<delay>]]
```

addr	This is the address of the sector (block) to unlock
delay	A delay time in milliseconds the BDI waits after sending the unlock command to the flash. For example, clearing all lock-bits of an Intel J3 Strata flash takes up to 0.7 seconds.

If "unlock" is used without any parameter, all sectors in the erase list with the UNLOCK option are processed.

To clear all lock-bits of an Intel J3 Strata flash use for example:

```
BDI> unlock 0xFF000000 1000
```

To erase or unlock multiple, continuos flash sectors (blocks) of the same size, the following Telnet commands can be used:

```
ERASE  <addr> <step> <count>
UNLOCK <addr> <step> <count>
```

addr	This is the address of the first sector to erase or unlock.
step	This value is added to the last used address in order to get to the next sector. In other words, this is the size of one sector in bytes.
count	The number of sectors to erase or unlock.

The following example unlocks all 256 sectors of an Intel Strata flash (28F256K3) that is mapped to 0x00000000. In case there are two flash chips to get a 32bit system, double the "step" parameter.

```
BDI> unlock 0x00000000 0x20000 256
```

© Copyright 1997-2015 by ABATRON AG Switzerland

3.2.5 Part [REGS]

In order to make it easier to access target registers via the Telnet interface, the BDI can read in a register definition file. In this file, the user defines a name for the register and how the BDI should access it (e.g. as memory mapped, memory mapped with offset, ...). The name of the register definition file and information for different registers type has to be defined in the configuration file.
The register name, type, address/offset/number and size are defined in a separate register definition file. This way, you can create one register definition file for a specific target processor that can be used for all possible positions of the internal memory map. You only have to change one entry in the configuration file.

An entry in the register definition file has the following syntax:

```
name   type   addr   size
```

name — The name of the register (max. 15 characters)

type — The register type

GPR	General purpose register
SYS	System register
MM	Absolute direct memory mapped register
PMM	Like MM but with disabled MMU during the access
DMM1...DMM4	Relative direct memory mapped register
APB	APB memory mapped register
DBG	External debug register

addr — The address, offset or number of the register

size — The size (8, 16, 32) of the register, default is 32

The following entries are supported in the [REGS] part of the configuration file:

FILE filename — The name of the register definition file. This name is used to access the file via TFTP. The file is loaded once during BDI startup.

filename	the filename including the full path
Example:	FILE C:\bdi\regs\reg40400.def

DMMn base — This defines the base address of direct memory mapped registers. This base address is added to the individual offset of the register.

base	the base address
Example:	DMM1 0x01000

© Copyright 1997-2015 by ABATRON AG Switzerland

Example for a register definition:

Entry in the configuration file:

```
[REGS]
FILE            $regARMV8-EL2.def
```

The register definition file:

```
;  System Register Numbers :
;  +-------+-------+-------+-------+
;  | | | | | | | | | | | | | | | |
;  +-+-----+-------+-------+-----+-+
;  |o| op1 |  CRn  |  CRm  | op2 |-|
;  +-+-----+-------+-------+-----+-+
;   ^
;   low bit of op0
;
;name           type    addr            size
;-----------------------------------------
;
;
; System Registers
;
midr_el1        SYS     0x8000
ctr_el0         SYS     0xb002
mpidr_el1       SYS     0x800a          64
revidr_el1      SYS     0x800c
dczid_el0       SYS     0xb00e
....
;
; External Debug Registers
;
dbg_edesr       DBG     0x00020         32
dbg_edecr       DBG     0x00024         32
dbg_edwar       DBG     0x00030         64
dbg_edscr       DBG     0x00088         32
...
;
;
; Cross-trigger Interface Registers
;
cti_control     DBG     0x10000         32
cti_intack      DBG     0x10010         32
cti_appset      DBG     0x10014         32
cti_appclear    DBG     0x10018         32
cti_apppulse    DBG     0x1001C         32
cti_inen0       DBG     0x10020         32
cti_inen1       DBG     0x10024         32
cti_outen0      DBG     0x100A0         32
cti_outen1      DBG     0x100A4         32
cti_triginsta   DBG     0x10130         32
cti_trigoutsta  DBG     0x10134         32
cti_chinsta     DBG     0x10138         32
cti_choutsta    DBG     0x1013C         32
cti_gate        DBG     0x10140         32
cti_asicctl     DBG     0x10144         32
;
```

© Copyright 1997-2015 by ABATRON AG Switzerland

3.3 Debugging with GDB

Because the target agent runs within BDI, no debug support has to be linked to your application. There is also no need for any BDI specific changes in the application sources. Your application must be fully linked because no dynamic loading is supported.

3.3.1 Target setup

Target initialization may be done at two places. First with the BDI configuration file, second within the application. The setup in the configuration file must at least enable access to the target memory where the application will be loaded. Disable the watchdog and setting the CPU clock rate should also be done with the BDI configuration file. Application specific initializations like setting the timer rate are best located in the application startup sequence.

3.3.2 Connecting to the target

As soon as the target comes out of reset, BDI initializes it and loads your application code. If RUN is selected, the application is immediately started, otherwise only the target PC is set. BDI now waits for GDB request from the debugger running on the host.

After starting the debugger, it must be connected to the remote target. This can be done with the following command at the GDB prompt:

```
(gdb)target remote bdi2000:2001
```

bdi2000	This stands for an IP address. The HOST file must have an appropriate entry. You may also use an IP address in the form xxx.xxx.xxx.xxx
2001	This is the TCP port used to communicate with the BDI

If not already suspended, this stops the execution of application code and the target CPU changes to background debug mode.

Remember, every time the application is suspended, the target CPU is freezed. During this time no hardware interrupts will be processed.

Note: For convenience, the GDB detach command triggers a target reset sequence in the BDI.

```
(gdb)...
(gdb)detach
... Wait until BDI has resetet the target and reloaded the image
(gdb)target remote bdi2000:2001
```

© Copyright 1997-2015 by ABATRON AG Switzerland

3.3.3 Breakpoint Handling

There are two breakpoint modes supported. One of them (SOFT) is implemented by replacing application code with a HLT instruction. The other (HARD) uses the built in breakpoint logic. If HARD is selected, only up to n breakpoints can be active at the same time.

The following example selects SOFT as the breakpoint mode:

```
BREAKMODE   SOFT      ;SOFT or HARD, HARD uses hardware breakpoints
```

The BDI supports only a GDB version that uses a Z-Packet to set breakpoints (GDB Version 5.0 or newer). GDB tells the BDI to set / clear breakpoints with this special protocol unit. The BDI will respond to this request by replacing code in memory with the HLT instruction or by setting the appropriate hardware breakpoint.

3.3.4 GDB monitor command

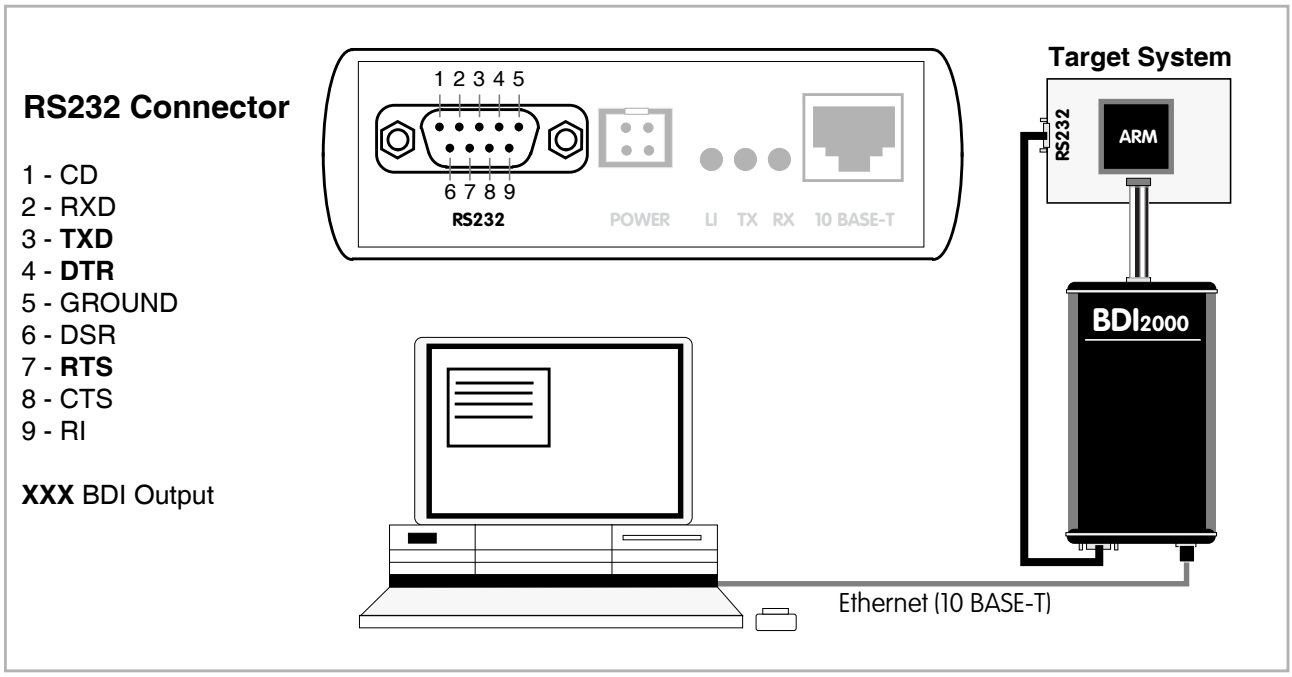
The BDI supports the GDB V5.x "monitor" command. Telnet commands are executed and the Telnet output is returned to GDB.

```
(gdb) target remote bdi2000:2001
Remote debugging using bdi2000:2001
0x10b2 in start ()
(gdb) monitor md 0 1
00000000 : 0xe59ff018  - 442503144  ...
```

© Copyright 1997-2015 by ABATRON AG Switzerland

3.3.5 Target serial I/O via BDI

A RS232 port of the target can be connected to the RS232 port of the BDI2000. This way it is possible to access the target's serial I/O via a TCP/IP channel. For example, you can connect a Telnet session to the appropriate BDI2000 port. Connecting GDB to a GDB server (stub) running on the target should also be possible.



The configuration parameter "SIO" is used to enable this serial I/O routing.
The BDI asserts RTS and DTR when a TCP connection is established.

```
[TARGET]
....
SIO    7      9600          ;Enable SIO via TCP port 7 at 9600 baud
```

Warning!!!
Once SIO is enabled, connecting with the setup tool to update the firmware will fail. In this case either disable SIO first or disconnect the BDI from the LAN while updating the firmware.

© Copyright 1997-2015 by ABATRON AG Switzerland

3.3.6 Target DCC I/O via BDI

It is possible to route a TCP/IP port to the ARM's debug communciation channel (DCC). This way, the application running on the target can output messages via DCC that are displayed for example in a Telnet window. The BDI routes every byte received via DCC to the connected TCP/IP channel and vice versa. Below some simple functions you can link to your application in order to implement IO via DCC.

```
#define MDCCSR_TX_FULL  (1L<<29)
#define MDCCSR_RX_FULL  (1L<<30)


static unsigned int read_dtr(void)
{
    unsigned int c;

    __asm__(
           "mrs x0,dbgdtrrx_el0\n"
           : "=r" (c));
    return c;
}

static void write_dtr(unsigned int c)
{
    __asm__(
           "msr dbgdtrtx_el0,x0\n"
           :
           : "r" (c));
}

static unsigned int read_mdccsr(void)
{
    unsigned int ret;

    __asm__(
           "mrs x0,mdccsr_el0\n"
           : "=r" (ret));
    return ret;
}

void write_dcc_char(unsigned int c)
{
    while(read_mdccsr() & MDCCSR_TX_FULL);
    write_dtr(c);
}

unsigned int read_dcc_char(void)
{
    while(!(read_mdccsr() & MDCCSR_RX_FULL));
    return read_dtr();
}


void write_dcc_string(const char* s)
{
    while (*s) write_dcc_char(*s++);
}
```

© Copyright 1997-2015 by ABATRON AG Switzerland

3.4 Telnet Interface

A Telnet server is integrated within the BDI. The Telnet channel is used by the BDI to output error messages and other information. Also some basic debug tasks may be done by using this interface. Enter help at the Telnet command prompt to get a list of the available commands.

Telnet Debug features:

- Display and modify memory locations
- Display and modify registers
- Single step a code sequence
- Set hardware breakpoints (for code and data accesses)
- Load a code file from any host
- Start / Stop program execution
- Programming and Erasing Flash memory

During debugging with GDB, the Telnet is mainly used to reboot the target (generate a hardware reset and reload the application code). It may be also useful during the first installation of the bdiGDB system or in case of special debug needs.

Multiple commands separated by a semicolon can be entered on one line.

Example of a Telnet session:

```
XGENE#0>info
    Core number       : 0
    Core state        : debug (AArch64 EL2)
    Debug entry cause : External Debug Request
    Current PC        : 0x000000001d00c2a4
    Current CPSR      : 0x800003c9 (EL2h)
XGENE#0>md 0x1d000000
1d000000 : d503201f d503201f d503201f d503201f  . ... ... ... ..
1d000010 : d503201f d503201f d503201f d503201f  . ... ... ... ..
1d000020 : d503201f d503201f d503201f d503201f  . ... ... ... ..
1d000030 : d503201f d503201f d503201f 17fffffb  . ... ... ......
1d000040 : d53800a0 d51c00a0 d51cc01f d2810005  ..8.............
              ......................

XGENE#0>ti
    Core number       : 0
    Core state        : debug (AArch64 EL2)
    Debug entry cause : Single Step
    Current PC        : 0x000000001d00c2a8
    Current CPSR      : 0x800003c9 (EL2h)
XGENE#0>
```

Notes:
The DUMP command uses TFTP to write a binary image to a host file. Writing via TFTP on a Linux/Unix system is only possible if the file already exists and has public write access. Use "man tftpd" to get more information about the TFTP server on your host.

© Copyright 1997-2015 by ABATRON AG Switzerland

3.4.1 Command list

```
"MD    [<address>] [<count>]  display target memory as word (32bit)",
"MDH   [<address>] [<count>]  display target memory as half word (16bit)",
"MDB   [<address>] [<count>]  display target memory as byte (8bit)",
"DUMP  <addr> <size> [<file>] dump target memory to a file",

"MM    <addr> <value> [<cnt>] modify word(s) (32bit) in target memory",
"MMH   <addr> <value> [<cnt>] modify half word(s) (16bit) in target memory",
"MMB   <addr> <value> [<cnt>] modify byte(s) (8bit) in target memory",
"MT    <addr> <count>         memory test",
"MC    [<address>] [<count>]  calculates a checksum over a memory range",
"MV                           verifies the last calculated checksum",

"RD    [<name>]               display general purpose or user defined register",
"RDUMP [<file>]               dump all user defined register to a file",
"RDALL                        display all ARM registers ",
"RDSYS <number>               display system register",
"RDFP                         display floating point register",
"RDVR                         display vector registers",

"RM    {<nbr>|<name>} <value> modify general purpose or user defined register",
"RMSYS <number> <value>       modify system register",
"RMFP  <number> <value>       modify floating point register",
"RMVR  <nbr><val val val val> modify vector register (four 32bit values)",
"MMU   {ENABLE | DISABLE}     enable / disable MMU via control register",

"DCPS  [{1 | 2 | 3}]          debug change/restore processor state",
"EXEC  <inst> [<r0> [<r1>]]   execute an instruction",
"                   <inst> : numeric coding or ICIALLU or ICIVAU or DCCVAU",
"RESET [HALT | RUN [time]]    reset the target system, change startup mode",
"GO    [<pc>]                 set PC and start current core",
"CONT  <cores>                restart multiple cores (<cores> = core bit map)",
"TI    [<pc>]                 single step an instruction",
"HALT  [<cores>]              force core(s) to debug mode (<cores> = core bit map)",

"BI    <addr>                 set instruction breakpoint",
"CI    [<id>]                 clear instruction breakpoint(s)",
"BD    [R|W] <addr>           set data watchpoint (32bit access)",
"BDH   [R|W] <addr>           set data watchpoint (16bit access)",
"BDB   [R|W] <addr>           set data watchpoint ( 8bit access)",
"BDM   [R|W] <addr> [<mask>]  set data watchpoint with address mask",
"CD    [<id>]                 clear data watchpoint(s)",

"INTDIS                       disable target interrupts while running",
"INTENA                       enable  target interrupts while running (default)",
"INFO                         display information about the current state",
"STATE                        display information about all cores",
```

© Copyright 1997-2015 by ABATRON AG Switzerland

The Telnet commands (cont.):

```
"LOAD   [<offset>] [<file> [<format>]] load program file to target memory",
"VERIFY [<offset>] [<file> [<format>]] verify a program file to target memory",
"PROG   [<offset>] [<file> [<format>]] program flash memory",
"                           <format>  : SREC, BIN, or ELF",
"ERASE  [<address> [<mode>]]  erase a flash memory sector, chip or block",
"                   <mode>  : CHIP, BLOCK or SECTOR (default is sector)",
"ERASE  <addr> <step> <count> erase multiple flash sectors",
"UNLOCK [<addr> [<delay>]]    unlock a flash sector",
"UNLOCK <addr> <step> <count> unlock multiple flash sectors",
"FLASH  <type> <size> <bus>   change flash configuration",
"FENA   <addr> <size>         enable autoamtic programming to flash memory",
"FDIS                         disable autoamtic programming to flash memory",

"DELAY  <ms>                  delay for a number of milliseconds",
"MEMACC {CORE | AHB [<hprot>]}select memory access mode",
"SELECT <core>                change the current core",
"ATTACH [<core>]              connect to a core",
"DETACH [<core>]              disconnect from a core",
"HOST   <ip>                  change IP address of program file host",
"PROMPT <string>              defines a new prompt string",

"QUERY [<core>]               display target configuration",
"CONFIG                       display or update BDI configuration",
"CONFIG <file> [<hostIP> [<bdiIP> [<gateway> [<mask>]]]]",
"HELP                         display command list",
"BOOT  [loader]               reboot the BDI and reload the configuration",
"QUIT                         terminate the Telnet session",



"-----------------------------------------",
"Low level access to CoreSight debug system:",
"-----------------------------------------",
"RDP   <addr>                 display Debug Port (DP) register",
"RAP   <addr>                 display Access Port (AP) register",
"RDBG  <addr> [<cnt>]         display core debug register",
"WDP   <addr> <value>         modify Debug Port (DP) register",
"WAP   <addr> <value>         modify Access Port (AP) register",
"WDBG  <addr> <value>         modify core debug register",
"MDAPB <addr> [<cnt>]         display APB memory",
"MMAPB <addr> <value>         modify  APB memory",
"MDAHB <addr> [<cnt>]         display AHB memory (32-bit)",
"MMAHB <addr> <value>         modify  AHB memory (32-bit)",
"MDAXI <addr> [<cnt>]         display AXI memory (64-bit)",
"MMAXI <addr> <value>         modify  AXI memory (64-bit)"
```

© Copyright 1997-2015 by ABATRON AG Switzerland

Additional info about some Telnet commands:

rdsys / rmsys:
The number for accessing system registers via "rdsys" or "rmsys" has to be built as follows:

```
+-------+-------+-------+-------+
| | | | | | | | | | | | | | | |
+-+-----+-------+-------+-----+-+
|o| op1 |  CRn  |  CRm  | op2 |-|
+-+-----+-------+-------+-----+-+
 ^
 low bit of op0
```

For examples look at the provided register definition files.

dcps:
This command allows to change the processor state in debug mode. With "dcps x" you can increase the exception level. A "dcps" without any parameter executes a DRPS instruction that restores the processor to the exception level and mode recorded in the current SPSR_ELn. Before executing the DRPS instruction the BDI loads the current SPSR_ELn with the locally stored CPSR value.

exec:
This allows to execute a single instruction in debug mode. The registers r0 and r1 are loaded with the provided values before the instruction is executed. Then the registers r0 and r1 are read back and displayed. Be aware that not all instructions can be executed in Debug state. The following example reads a 64-bit value from 0x1d000000 via "ldr x1,[x0],#8":

```
XGENE#0>exec 0xf8408401 0x1d000000
r0: 0x000000001d000008 r1: 0xd503201f14000010
```

For some cache maintenance instructions a name can be entered instead of the numeric opcode:

```
XGENE#0>exec iciallu
XGENE#0>exec icivau 0x000000001d00c2a8
XGENE#0>exec dccvau 0x000000001d0e9f68
```

© Copyright 1997-2015 by ABATRON AG Switzerland

3.5 Multi-Core Support

The bdiGDB system supports concurrent debugging of up to 8 cores. Via Telnet you can switch between the cores with the command "select <0..7>". In the configuration file, simply begin the line with the appropriate core number. If there is no #n in front of a line, the BDI assumes core #0.

Up to 8 cores can be prepared for GDB debugging. To select a core for GDB debugging define an approriate debug port number. Only core #0 has a default debug port of 2001 assigned. For every selected core you can start its own individual GDB session.

The following example defines 4 cores for debugging. For a complete example, look at the configuration examples.

```
[TARGET]
POWERUP         3000                    ;start delay after power-up detected in ms
CLOCK           8000000                 ;JTAG clock 8 MHz
TRST            OPENDRAIN               ;TRST driver type (OPENDRAIN | PUSHPULL)
;
; CoreID#0 parameters (active core after reset)
#0 CPUTYPE      X-GENE 0xfc010000       ;X-Gene CPU 0
#0 STARTUP      HALT                    ;halt as soon as possible
#0 ENDIAN       LITTLE                  ;memory model (LITTLE | BIG)
#0 VECTOR       CATCH  RST OSU TDA      ;Reset and OS unlock catch, Trap SW access
#0 BREAKMODE    SOFT                    ;SOFT or HARD
#0 MEMACCESS    CORE   10               ;memory access via Core  (80 TCK's access delay)
;
; CoreID#1 parameters:
#1 CPUTYPE      X-GENE 0xfc110000       ;X-Gene CPU 1
#1 STARTUP      RUN
#1 ENDIAN       LITTLE
#1 BREAKMODE    HARD
#1 MEMACCESS    CORE   10
;
; CoreID#2 parameters:
#2 CPUTYPE      X-GENE 0xfc210000       ;X-Gene CPU 2
#2 STARTUP      RUN
#2 ENDIAN       LITTLE
#2 BREAKMODE    HARD
#2 MEMACCESS    CORE   10
;
; CoreID#3 parameters:
#3 CPUTYPE      X-GENE 0xfc310000       ;X-Gene CPU 3
#3 STARTUP      RUN
#3 ENDIAN       LITTLE
#3 BREAKMODE    HARD
#3 MEMACCESS    CORE   10
;
[HOST]
#0 PROMPT       XGENE#0>
#1 PROMPT       XGENE#1>
#2 PROMPT       XGENE#2>
#3 PROMPT       XGENE#3>
#0 DEBUGPORT    2800
#1 DEBUGPORT    2801
#2 DEBUGPORT    2802
#3 DEBUGPORT    2803
```

Note:
Be aware that via Telnet you select the core by its BDI core ID (#n). This BDI core ID is not necessary the core number within the SOC.

© Copyright 1997-2015 by ABATRON AG Switzerland

Multi-Core related Telnet commands:

STATE	Display information about all cores.
SELECT <core>	Change the current Telnet core
CONT <cores>	Restart one or multiple cores <cores> core bit map Example: cont 0x000d ; restart core #0, #2, #3
HALT [<cores>]	Force one or multiple cores to debug mode. If there is no <cores> parameter, the currently selected core is forced to debug mode. <cores> core bit map Example: halt 0x00ff ; halt 8 cores #0...#7

If a group of cores is restarted with the "cont" command then all cores in the group are started synchronously and CTI and CTM is setup so that all cores in this group halt when one of it halts.

If there is a core bit map entered for the "halt" command then all cores in the bit map are halted synchronously.

Example where all cores halt when core#0 halts on a breakpoint:

```
XGENE#0>stat
Core#0: halted 0x00000043eff65900 EDBGRQ signal
Core#1: halted 0x000000000000022c EDBGRQ signal
Core#2: halted 0x000000000000022c EDBGRQ signal
Core#3: halted 0x000000000000022c EDBGRQ signal
Core#4: halted 0x000000000000022c EDBGRQ signal
Core#5: halted 0x000000000000022c EDBGRQ signal
Core#6: halted 0x000000000000022c EDBGRQ signal
Core#7: halted 0x000000000000022c EDBGRQ signal

XGENE#0>bi 0x00000043eff65900
Breakpoint identification is 0

XGENE#0>cont 0xff
- TARGET: core #0 has entered debug mode
- TARGET: core #1 has entered debug mode
- TARGET: core #2 has entered debug mode
- TARGET: core #3 has entered debug mode
- TARGET: core #4 has entered debug mode
- TARGET: core #5 has entered debug mode
- TARGET: core #6 has entered debug mode
- TARGET: core #7 has entered debug mode

XGENE#0>stat
Core#0: halted 0x00000043eff65900 Breakpoint
Core#1: halted 0x000000000000022c EDBGRQ signal
Core#2: halted 0x000000000000022c EDBGRQ signal
Core#3: halted 0x000000000000022c EDBGRQ signal
Core#4: halted 0x000000000000022c EDBGRQ signal
Core#5: halted 0x000000000000022c EDBGRQ signal
Core#6: halted 0x000000000000022c EDBGRQ signal
Core#7: halted 0x000000000000022c EDBGRQ signal
```

© Copyright 1997-2015 by ABATRON AG Switzerland

Multi-Core Restart via GDB continue:

Then core specific parameter CTI allows to define a group of cores that should be restarted when GDB sends the "continue" command to the BDI. This has the same effect as the Telnet "cont" command. Via the cgroup option you define what the BDI does in response to the GDB continue command:

- If there is no CGROUP defined then the core is restarted as usual.
- If the CGROUP core mask defines only the actual core then this core is prepared for restart but the final step to actually restart is made pending. To actually restart it a "continue" command from the master GDB session (see next) or the Telnet "cont" command is necessary.
- If the CGROUP core mask includes other cores beside the actual one, then all cores in the mask are prepared for restart (if not already done) and finally the whole core group is restarted at the same time and CTI and CTM is setup so that all cores in this group halt when one of it halts.

This supports two different debug scenarios where the first one is actually a special case of the second one:

- Debug only one core via GDB but make sure that always all cores are either halted or running. For this only one CGROUP for the debugged core is necessary. The core mask defines all the cores.
- Debug multiple cores (not necessary all cores) with different GDB sessions. Here one core will be let's say the master core with the attached master GDB session. Always continue all other GDB session (cores) before entering the continue command in the master GDB session. For the master core define the CGROUP mask with all cores. For other cores set only the bit in the core mask that represents the core itself.

© Copyright 1997-2015 by ABATRON AG Switzerland

4 Specifications

Operating Voltage Limiting	5 VDC ± 0.25 V
Power Supply Current	typ. 500 mA max. 1000 mA
RS232 Interface: Baud Rates	9’600,19’200, 38’400, 57’600,115’200
Data Bits	8
Parity Bits	none
Stop Bits	1
Network Interface	10 BASE-T
Serial Transfer Rate between BDI and Target	up to 16 Mbit/s
Supported target voltage	1.8 – 5.0 V (3.0 – 5.0 V with Rev. A/B)
Operating Temperature	+ 5 °C ... +60 °C
Storage Temperature	-20 °C ... +65 °C
Relative Humidity (noncondensing)	<90 %rF
Size	190 x 110 x 35 mm
Weight (without cables)	420 g
Host Cable length (RS232)	2.5 m

Specifications subject to change without notice

© Copyright 1997-2015 by ABATRON AG Switzerland

5 Environmental notice

Disposal of the equipment must be carried out at a designated disposal site.

6 Declaration of Conformity (CE)

CE

DECLARATION OF CONFORMITY

This declaration is valid for following product:

Type of device: BDM/JTAG Interface
Product name: BDI2000

The signing authorities state, that the above mentioned equipment meets the requirements for emission and immunity according to

EMC Directive 89/336/EEC

The evaluation procedure of conformity was assured according to the following standards:

EN 50081-2
EN 50082-2

This declaration of conformity is based on the test report no. QNL-E853-05-8-a of QUINEL, Zug, accredited according to EN 45001.

Manufacturer:

ABATRON AG
Stöckenstrasse 4
CH-6221 Rickenbach

Authority:

Max Vock
Marketing Director

Ruedi Dummermuth
Technical Director

Rickenbach, May 30, 1998

© Copyright 1997-2015 by ABATRON AG Switzerland

7 Abatron Warranty and Support Terms

7.1 Hardware

ABATRON Switzerland warrants that the Hardware shall be free from defects in material and workmanship for a period of 3 years following the date of purchase when used under normal conditions. Failure in handling which leads to defects or any self-made repair attempts are not covered under this warranty. In the event of notification within the warranty period of defects in material or workmanship, ABATRON will repair or replace the defective hardware. The customer must contact the distributor or Abatron for a RMA number prior to returning.

7.2 Software

License

Against payment of a license fee the client receives a usage license for this software product, which is not exclusive and cannot be transferred.

Copies

The client is entitled to make copies according to the number of licenses purchased. Copies exceeding this number are allowed for storage purposes as a replacement for defective storage mediums.

Update and Support

The agreement includes free software maintenance (update and support) for one year from date of purchase. After this period the client may purchase software maintenance for an additional year.

7.3 Warranty and Disclaimer

ABATRON AND ITS SUPPLIERS HEREBY DISCLAIMS AND EXCLUDES, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT.

7.4 Limitation of Liability

IN NO EVENT SHALL ABATRON OR ITS SUPPLIERS BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING, WITHOUT LIMITATION, ANY SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE HARDWARE AND/OR SOFTWARE, INCLUDING WITHOUT LIMITATION, LOSS OF PROFITS, BUSINESS, DATA, GOODWILL, OR ANTICIPATED SAVINGS, EVEN IF ADVISED OF THE POSSIBILITY OF THOSE DAMAGES.

The hardware and software product with all its parts, copyrights and any other rights remain in possession of ABATRON. Any dispute, which may arise in connection with the present agreement shall be submitted to Swiss Law in the Court of Zug (Switzerland) to which both parties hereby assign competence.

© Copyright 1997-2015 by ABATRON AG Switzerland

Appendices

A Troubleshooting

Problem
The firmware can not be loaded.

Possible reasons

- The BDI is not correctly connected with the target system (see chapter 2).
- The power supply of the target system is switched off or not in operating range (4.75 VDC ... 5.25 VDC) --> MODE LED is OFF or RED
- The built in fuse is damaged --> MODE LED is OFF
- The BDI is not correctly connected with the Host (see chapter 2).
- A wrong communication port (Com 1...Com 4) is selected.

Problem
No working with the target system (loading firmware is ok).

Possible reasons

- Wrong pin assignment (BDM/JTAG connector) of the target system (see chapter 2).
- Target system initialization is not correctly --> enter an appropriate target initialization list.
- An incorrect IP address was entered (BDI2000 configuration)
- BDM/JTAG signals from the target system are not correctly (short-circuit, break, ...).
- The target system is damaged.

Problem
Network processes do not function (loading the firmware was successful)

Possible reasons

- The BDI2000 is not connected or not correctly connected to the network (LAN cable or media converter)
- An incorrect IP address was entered (BDI2000 configuration)

© Copyright 1997-2015 by ABATRON AG Switzerland

B Maintenance

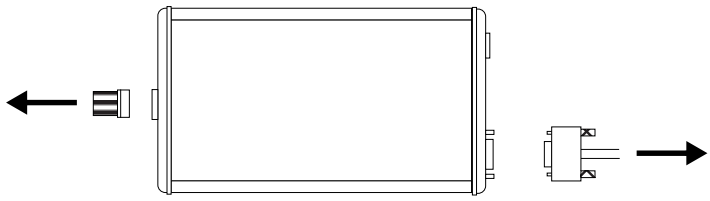
The BDI needs no special maintenance. Clean the housing with a mild detergent only. Solvents such as gasoline may damage it.

If the BDI is connected correctly and it is still not responding, then the built in fuse might be damaged (in cases where the device was used with wrong supply voltage or wrong polarity). To exchange the fuse or to perform special initialization, please proceed according to the following steps:

Observe precautions for handling (Electrostatic sensitive device)
Unplug the cables before opening the cover.
Use exact fuse replacement (Microfuse MSF 1.6 AF).

1

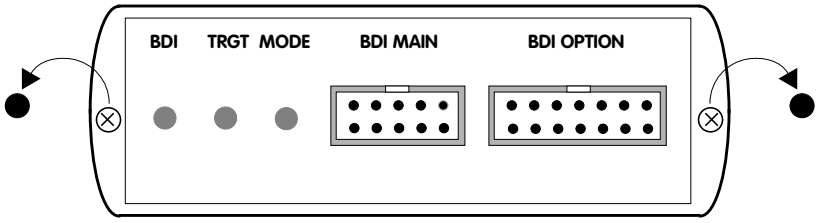
1.1 Unplug the cables



2

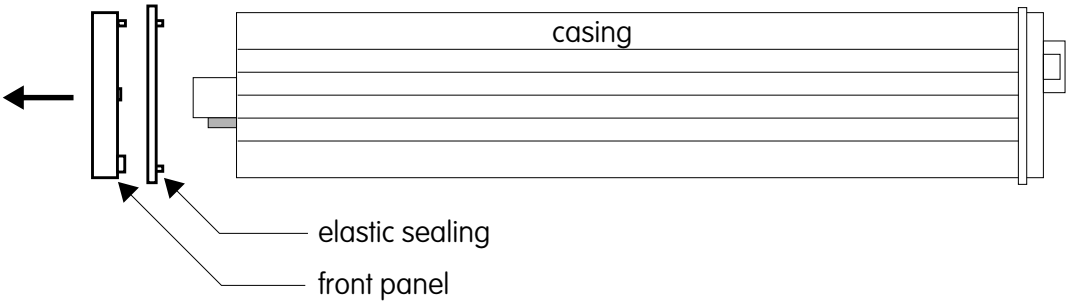
2.1 Remove the two plastic caps that cover the screws on target front side (e.g. with a small knife)

2.2 Remove the two screws that hold the front panel



3

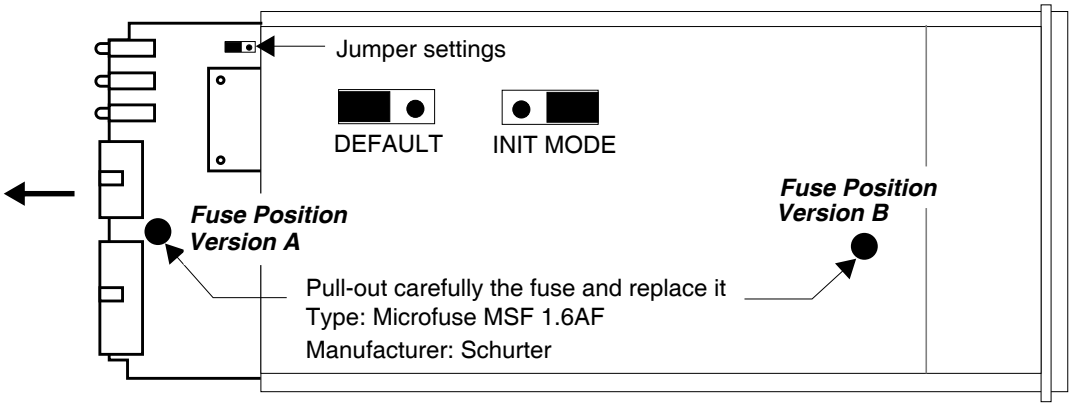
3.1 While holding the casing, remove the front panel and the red elastig sealing



© Copyright 1997-2015 by ABATRON AG Switzerland

4

4.1 While holding the casing, slide carefully the print in position as shown in figure below

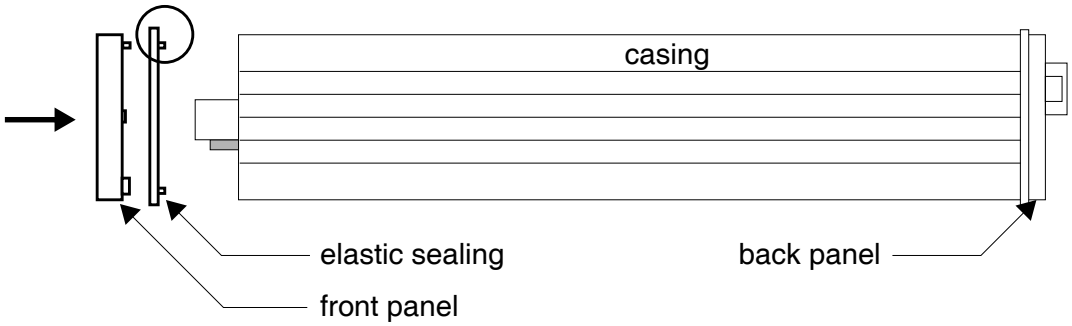


5

Reinstallation

5.1 Slide back carefully the print. Check that the LEDs align with the holes in the back panel.

5.2 Push carefully the front panel and the red elastig sealing on the casing. Check that the LEDs align with the holes in the front panel and that the position of the sealing is as shown in the figure below.



5.3 Mount the screws (do not overtighten it)

5.4 Mount the two plastic caps that cover the screws

5.5 Plug the cables

Observe precautions for handling (Electrostatic sensitive device)
Unplug the cables before opening the cover.
Use exact fuse replacement (Microfuse MSF 1.6 AF).

© Copyright 1997-2015 by ABATRON AG Switzerland

C Trademarks

All trademarks are property of their respective holders.

© Copyright 1997-2015 by ABATRON AG Switzerland